Technical Report 1411

Robust Photo-topography by Fusing Shape-from-Shading and Stereo

Clay Matthew Thompson

MIT Artificial Intelligence Laboratory

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE February 1993	3. REPORT TYPE AND DATES COVERED technical report

4. TITLE AND SUBTITLE Robust Photo-topography by Fusing Shape-from-Shading and Stereo	5 FUNDING NUMBERS N00014-91-J-4038 NAS5-31352
6. AUTHOR(S) Clay Matthew Thompson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139	8 PERFORMING ORGANIZATION REPORT NUMBER AI-TR 1411
9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Information Systems Arlington, Virginia 22217	10. SPONSORING MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution of this document is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Methods for fusing two computer vision methods are discussed and several example algorithms are presented to illustrate the variational method of fusing algorithms. The example algorithms solve the ***photo-topography problem***; that is, the algorithms seek to determine planet topography given two images taken from two different locations with two different lighting conditions. The algorithms each employ a single cost function that combines the computer vision methods of shape-from-shading and stereo in different ways. The algorithms are closely coupled and take into account all the constraints of the photo-topography problem.

One such algorithm, the *z*-only algorithm, can accurately and robustly estimate the height of a surface from two given images. Results of running the algorithms on four synthetic test image sets of varying difficulty are presented.

14. SUBJECT TERMS (key words) shape-from-shading matlab stereo computer vision fusion	15. NUMBER OF PAGES 169
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18 SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNCLASSIFIED

Robust Photo-topography by Fusing Shape-from-Shading and Stereo

by

Clay Matthew Thompson

B.S.A.A., University of Washington (1983)
M.S., Stanford University (1984)

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1993

Robust Photo-topography by Fusing Shape-from-Shading and Stereo

by
Clay Matthew Thompson

Submitted to the Department of Mechanical Engineering
on November 9, 1992, in par   llment of the
requirements for t   of
Doctor of Philo

Abstract

Methods for fusing two computer vision methods are discussed and several example algorithms are presented to illustrate the variational method of fusing algorithms. The example algorithms solve the ***photo-topography problem***; that is, the algorithms seek to determine planet topography given two images taken from two different locations with two different lighting conditions. The algorithms each employ a single cost function that combines the computer vision methods of shape-from-shading and stereo in different ways. The algorithms are closely coupled and take into account all the constraints of the photo-topography problem.

One such algorithm, the z-only algorithm, can accurately and robustly estimate the height of a surface from two given images. Results of running the algorithms on four synthetic test image sets of varying difficulty are presented.

Thesis Supervisor: Berthold K. P. Horn
Title: Professor, Electrical Engineering and Computer Science

We know of no universe, whatsoever,
where music is not in the center of the rocks.
— Brian Swimme, *Canticle to the Cosmos* (1990)

Acknowledgments

I would like to thank my advisor, Professor Berthold Horn, for providing guidance and support, and for suggesting the topic which became this thesis. I found working with him on this project to be very enjoyable. I am also grateful to my committee members, Professor Derek Rowell and Professor Alan Willsky, for their contribution to the success of this thesis.

I would like to thank John N. Little of The MathWorks, Inc. for providing me with alpha and beta copies of MATLAB 4.0 which greatly simplified the implementation and analysis of the algorithms presented in this thesis. I would also like to thank Mike Caplinger for providing the Viking images of Mars.

I am particularly grateful for the emotional support I have received from my parents over the years. In many ways, this Ph.D. is also theirs.

Lastly, I would like to thank my girlfriend Susan for her vision, understanding, and colorful imagery that continues to have a positive impact in my life.

This research was supported in part by NASA contract NAS5-31352. Additional support came from DARPA contract N00014-91-J-4038 which helps support general activities at the M.I.T. Artificial Intelligence Laboratory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the years computer vision researchers have developed algorithms to interpret gray scale and color images. These algorithms can be used to extract various types of information from the images such as relative motion between images (motion-vision), depth from a pair of images (binocular stereo), surface orientation from a set of images with different lighting (photometric stereo), surface orientation from a single image (shape-from-shading and other shape-from-X algorithms). Most of these algorithms work well in specialized circumstances but not as well in general. They are not very robust. They work well if everything is perfect, but don't do very well in sub-optimal conditions. They operate with many restrictive assumptions, which limit their applicability to many real problems. A way is needed to create more robust methods that can be applied to wide range of input images.

One obvious approach is to research ways of enhancing the existing methods to make them applicable to a wider range of inputs. Another approach is to combine one or more methods to produce a hybrid method with better, more robust performance. This thesis looks into the latter approach. The resulting algorithms are called *fused algorithms*. Fusing of two or more computer vision methods, if they bring different information to the problem, can create a more robust solution. It's like Kalman filtering: with more sensors you get a better estimate.

This thesis explores and answers the following questions:

1. What are the different fusion paradigms and how do they differ?

2. Which fusion paradigm are best for vision problems?

3. How much additional performance and robustness can be obtained via fusion?

There are two main approaches to fusing disparate algorithms. I call them the ***module-based approach*** and the *variational approach*. In the module-based approach, the fused algorithms are run separately on the input images. The various outputs, (they might be a sparse depth map from binocular stereo and a dense surface orientation from shape-from-shading), are then combined somehow to generate a single

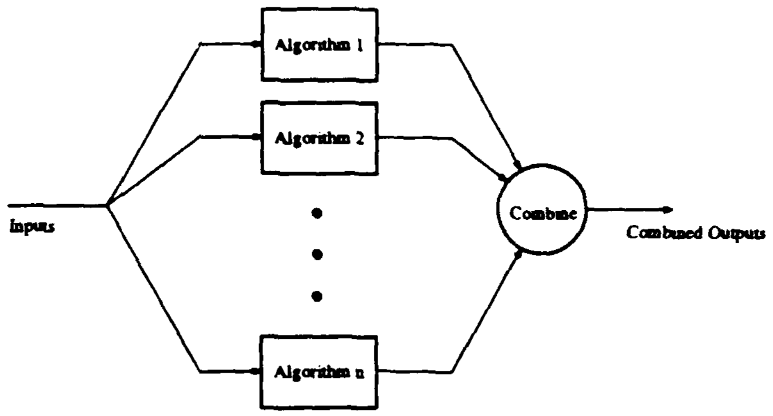


Figure 1-1: Generic module-based fusion method flow chart.

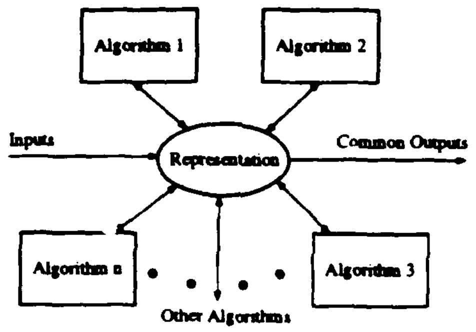


Figure 1-2: Generic variational-based fusion method flow chart.

solution (see Figure 1-1). The module-based approach is easy to implement since existing algorithms can be put together in an ad-hoc manner without having to know much about the inner workings of each algorithm. The outputs are then combined using physically derived constraints to generate the desired combined solution. The approach can be applied successfully, but has the disadvantage that this approach doesn't fully exploit the information coupling between the methods.

The variational approach, on the other hand, closely couples the algorithms together (see Figure 1-2). This is achieved by formulating a combined cost function based on the cost functions of the separate algorithms to be fused. The result is a combined optimization problem which takes into account both the explicit and implicit constraints between the methods. Variational methods, by their nature, can exploit any orthogonalities (information content) in the methods. By exploiting all the information, the variational approach has the potential to create robust, well performing combinations of algorithms which can be applied to a wide range of input images. How to create such algorithms is the focus of this thesis.

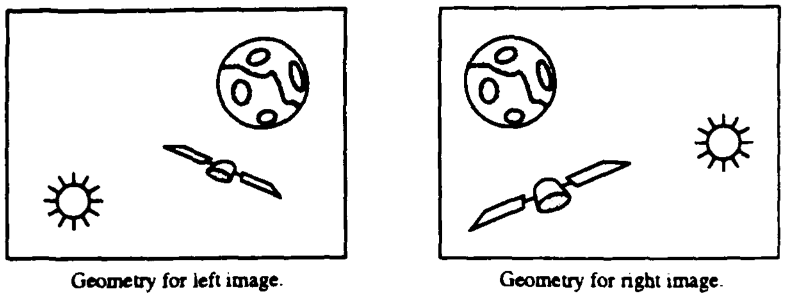


Figure 1-3: Example camera and light source geometry for the two images of photo-topography.

1.1 Photo-topography

A problem that can benefit greatly from a fused vision algorithm is the problem of photo-topography. Photo-topography seeks to determine the topography of a planet's surface based on two images of the planet, taken from two different vantage points at two different times. This situation is illustrated in Figure 1-3.

In some ways, this situation is analogous to binocular stereo. Unfortunately, the images are typically taken with two different light source positions as shown in the figure. The two images that result will, in general, look quite different from each other even though they are images of the same surface patch. One such set of images is shown in Figure 9-1 on page 121.

Contrast this with the situation that is normally true for binocular stereo images. Stereo image pairs are usually taken simultaneously (or nearly simultaneously), from positions that are near to each other, and with the same lighting. The images that result look very similar to each other except for the relative shift of objects (i.e. the disparity) due to their distance from the cameras. If the disparity for all points in the images and the relative geometry of the cameras is known, then the depth of the objects can be computed directly. The images are similar, so most stereo algorithms determine the disparity by trying to match features in one image with features in the other. Since the images from photo-topography look different from each other, this matching approach doesn't work.

Photo-topography also shares some aspects with a shape-from-shading problem. Shape-from-shading takes a gray-scale image of a surface and determines the surface topography by exploiting the shading information in the image. Shape-from-shading requires that the surface reflectance properties be known. Assuming the reflectance properties of the planet's surface are known, we could use a shape-from-shading algorithm to estimate the surface topography from each of the photo-topography images. Unfortunately, the surface estimates based on each image will typically be different. They may not even be very similar. In the worse case, the surface estimates from each image may not have the same orientation; one could be concave while the other is convex. Thus planet photo-topography is a perfect problem on which to test a fusion algorithm based on shape-from-shading and stereo.

Photo-topography also has aspects in common with a photometric stereo problem. Photometric stereo uses two images of the same scene taken with two different lighting conditions. Both images are from the same vantage point: the camera is not moved between images. The result is two images that look different but where the correspondence is explicitly known (a position in one image is always matched with the same position in the other image). If the light positions are far enough apart, it is possible to determine the surface orientation directly. For Lambertian reflectance, two images can constrain the surface orientation to two possible values at each point. With three images, it is possible to find a unique surface orientation at each point.

The photo-topography images have two different light source positions, but the correspondence is based on binocular stereo. Like photometric stereo, the two light sources can constrain the surface orientation, but only if the correspondence is known. Thus in photo-topography, photometric stereo and binocular stereo are closely linked.

Currently, photo-topography problems are solved using a characteristic strips method [Davis and Soderblom, 1984] or more modern shape-from-shading methods [Van Hove and Carlotto, 1986]. The characteristic strips method uses a general method of solving partial differential equations and is similar to some early shape-from-shading algorithms. In this method, the solution is found along strips starting from a given (or known) point. The solutions obtained are local, and may not cover the space of interest.

1.2 Related Research

This research is most closely related to the work of Horn [Ikeuchi and Horn, 1981], [Horn and Brooks, 1986], [Horn, 1986], [Horn, 1989], Gennert [Gennert, 1987], and Szeliski [Szeliski, 1990], [Szeliski, 1991]. The variational (least squares) approach I have taken is based significantly on the work of Horn [Horn, 1989], [Horn and Schunk, 1981], [Negahdaripour and Horn, 1985] and on insights gained from my background in control/estimation. The shape-from-shading part of this thesis builds upon the work of Horn [Horn, 1989], Szeliski [Szeliski, 1991], and Leclerc and Bobick [Leclerc and Bobick, 1991]. The stereo part of this thesis builds on the gray-scale stereo algorithm of Gennert [Gennert, 1987]. I use the hierarchical basis functions of Szeliski and use conjugate gradient optimization, as do Leclerc and Bobick [Leclerc and Bobick, 1991].

This research is also related to the work of Hartt and Carlotto [Hartt and Carlotto, 1989], [Heipke, 1992], Wildey [Wildey, 1973], and McEwen [McEwen, 1985] in that they try to solve the same problem; that is, the determination of planet topography from a pair of images.

Hartt and Carlotto use a Bayesian formulation with multiple images that in some ways is very similar to the methods I use. Their cost function and their approach is however very different. The Bayesian formulation leads them to a cost function of the form (using my notation)

$$U(z) = \iint \sum_k |E^{(k)}(x,y) - Rk(p,q)| + \lambda |\nabla z(x,y)|^2 dx\, dy. \tag{1.1}$$

While they discuss methods of estimating the albedo of the surface, results are shown for constant albedo surfaces with Lambertian reflectance.[1]

They solve their problem using a Markov Random Field Model via a Gibbs distribution [Geman and Geman, 1984]. They use the Metropolis optimization algorithm [Metropolis *et al.*, 1953] with a coarse-to-fine multi-resolution strategy to obtain their results. They show results for the algorithm on multiple (2) images based on a digital elevation map of upstate New York. They obtain better estimates when two images are used instead of one. It is hard to determine from the results they show, but it appears that their estimates contain quite a bit of error (only the estimated images are shown). The efficiency of their algorithm is also hard to gauge. They state in their paper that approximately 200 sweeps of each image is performed at each resolution. Inferring that they used 60-by-60 pixel images, with 3 resolution levels, and each sweep evaluated the cost function n times where n is the number of pixels on that level, the number of function evaluations they required is approximately

$$\begin{aligned} N &\approx 200 * (60^2 + 30^2 + 15^2) && (1.2) \\ &\approx 945000 \text{ function evaluations.} && (1.3) \end{aligned}$$

This is significantly more function evaluations that the algorithms I have developed (my algorithms typically require at most 1500 function evaluations for 65-by-65 pixel images and much less on some images).

Heipke discusses the solution to a multiple image photo-topography problem. In the paper [Heipke, 1992], he states that he uses a least squares cost function. In addition, the constraint equations that he develops are the same as mine except that they apply to more than two images. Since Heipke is not very explicit in this paper, it is difficult to tell exactly how he solves the problem. His results are encouraging but also show that his method is very sensitive to noise in the images. Heipke's algorithm shares many things in common with the algorithms I develop in this thesis since he starts with the same constraint equations. Even so, his algorithm doesn't perform as well as my algorithms.

The work of McEwen is based on traditional photoclinometry methods and is thus profile based. In [McEwen, 1985], McEwen uses information from two images of the same location under different lighting conditions to distinguish between albedo and reflectance effects. The image sets McEwen uses are taken from nearly the same location so that the pixel correspondence is easy. In many ways, McEwen's methods parallel photometric stereo much more than my algorithms.

[1] I make similar assumptions for most of this thesis.

Wildey basically solves a stereo problem [Wildey, 1973]. He assumes that that the light source is in the same position for each image and tries to determine pixel correspondence by matching image brightness. His method estimates the topography of the planet surface as it proceeds. He mentions an enhancement to this method that would take into account the estimated brightness of the reconstructed surface. This enhancement is closely related to my algorithms (and shape-from-shading) and would be applicable to images with similar lighting.

Another related paper is the shape-from-shading and stereo fusion algorithm of Grimson [Grimson, 1984]. In that paper, Grimson describes a method for enhancing the surface reconstruction step required after feature-based stereo using shading information. His method uses the shading information only along features to constrain the surface reconstruction and is loosely based on the methods of photometric stereo [Woodham, 1980].

1.2.1 Relation of this thesis to sensor fusion

The goal of this thesis is to investigate ways of combining or fusing two different computer vision methods. Thus this thesis is related to sensor fusion techniques (also called data fusion or information fusion in some contexts). Sensor fusion is a way of combining the information from multiple sensors to create better parameter estimates than can be achieved with the individual sensors alone.

Fusion has a much longer history than its use in vision algorithms. Researchers and engineers have been combining disparate sensors to obtain better estimates of the outside world for a long time. In the literature, three different approaches can be identified: 1) Fusion via estimation theory, 2) Fusion via decision theory, and 2) fusion via artificial intelligence methods.

The traditional approach involves the use of Kalman Filtering [Gelb, 1974]. In Kalman filtering, or estimation as it more broadly called, the sensor physics are analyzed and a noise model is postulated. Based on this information, a cost function is formulated. Typical cost functions are based on maximum likelihood estimation, maximum a priori estimation, minimum variance estimation, or least squares estimation. For simple noise models (such as Gaussian white noise), these cost functions have the form,

$$\min_x J = \int L(x, \ldots) \tag{1.4}$$

possibly subject to additional constraints imposed by the physics. $L(x, \ldots)$ is some non-linear functional. This same type of cost function is formed when solving vision problems using the variational approach [Horn, 1989]. Estimation theory-based fusion includes methods that rely on Bayesian statistics [Lee, 1990], [Richardson and Marsh, 1988], [Hung *et al.*, 1988], as well as least-squares approaches [Shaw *et al.*, 1988].

Decision theory approaches use Bayesian Reasoning [Thomopoulos, 1989], Shafer-Dempster Reasoning [Bogler, 1987] or ad hoc methods [Mitiche and Aggarwal, 1986].

The artificial intelligence methods range from rule-based hypotheses [Belknap *et al.*, 1986], [Belknap *et al.*, 1986], [Nandhakumar and Aggarwal, 1988], to blackboard schemes [Harmon *et al.*, 1986], to knowledge representation approaches [Pau, 1989], to artificial neural networks [Brown *et al.*, 1991]. Sensor fusion is an important problem for mobile robots and military threat detection as evidenced by the above references.

An important problem in sensor fusion that is not addressed in this thesis is the fusion of widely differing sensors. The problem is particularly difficult for mobile robots where the sensors may provide information at several conceptual levels. For example, sensors may be available for wheel rotation, range images, visual images, touch and sound. Combining information from such disparate sources is much more difficult than combining information from sources that operate on the same level (such as combining infrared, range and visual images).

1.2.2 Relation of this thesis to vision fusion schemes

The computer vision literature is roughly segmented along *module* boundaries. These modules are related to our current ideas about how human vision works. For instance there are algorithms for binocular stereo, motion vision, and a whole host of shape-from-X algorithms. Each of these algorithms performs reasonably well in ideal conditions but can degrade quickly in typical real-world situations. It has been suggested by many researchers [Aloimonos and Basu, 1988], [Waxman and Duncan, 1986] that a combination of these modules will do a better job of estimating the external world. Of course, these ideas are firmly supported by estimation theory where it can be easily shown that adding sensors cannot degrade an estimate and most often enhances it.

Along these lines, some researchers in the computer vision community have been looking into ways of combining more than one vision cue (and associated algorithm) in order to obtain either better or more robust estimates of the external world. The techniques for combining the methods range from those that use Bayesian estimation theory [Matthies and Elfes, 1988], [Hartt and Carlotto, 1989], to those that use module-based methods [Moerdler and Kender, 1987], [Moerdler and Boult, 1988], [Grimson, 1984]. Another common technique is to use an analytical approach [Waxman and Duncan, 1986], [Aloimonos and Basu, 1988], [Hu and Shrikhande, 1990], relying on the constraints from the fused cues (under particular assumptions) to generate either a unique or a finite number of possible solutions.

I prefer the estimation-based methods, since all the information available can be exploited, and the assumptions behind a particular method can be quantified. These methods also rest on a firm foundation. The module-based methods, in contrast, are more ad hoc. The analytical approach can be used within the estimation-based approach to constrain the solution.

1.2.3 Relation of this thesis to photoclinometry

One of the reasons for this research is to improve methods of planetary mapping. In particular, this research aims to widen the applicability of computer methods to this difficult task. Current techniques for planetary mapping work either on symmetric objects or along ridges [Davis and Soderblom, 1984], or use the same methods as shape-from-shading [Wildey, 1975], [Van Hove and Carlotto, 1986], [Kirk, 1987] although the theory was developed independently within the geophysics community. In that community, shape-from-shading is called *two-dimensional photoclinometry*. Important aspects for photoclinometry are the determination of the true reflectance function for a planet [Davis and McEwen, 1984], [Wilson *et al.*, 1985], [McEwen, 1991], [Helfenstein *et al.*, 1991] and how to deal with non-constant albedo [Davis and Soderblom, 1984], [Helfenstein *et al.*, 1991].

While there is no doubt that using the correct reflectance function and accounting for non-constant albedo is important, most of the algorithms in this thesis are for the simplified case of Lambertian reflectance and constant albedo. Both of these simplifications are not fatal: the algorithms can be easily generalized to other reflectance functions and the non-constant albedo case is dealt with in Section 8.1. In fact, the algorithms developed will work, without change, with any reflectance function that is smooth and that doesn't contain multiple extrema.

Chapter 2

Background

2.1 Coordinate Systems

The solution to a photo-topography problem is a description of the surface topography. The most straightforward description for the surface is to represent it as a height function over some suitable 2-D domain,[1] that is

$$z = z(x, y). \tag{2.1}$$

Within the vision literature, there are two different choices for the 2-D domain. The *image-centered domain* uses the coordinates of the image as the fundamental domain and assigns a depth (or height) value to the surface point that projects to each image position. If orthographic projection is used, then the projection mapping is straightforward. However if perspective projection is used, then this mapping can become quite complex (especially for surface normal calculations). The *object-centered domain* uses a coordinate system associated with the object and assigns a surface

[1]Also known as a Monge patch.

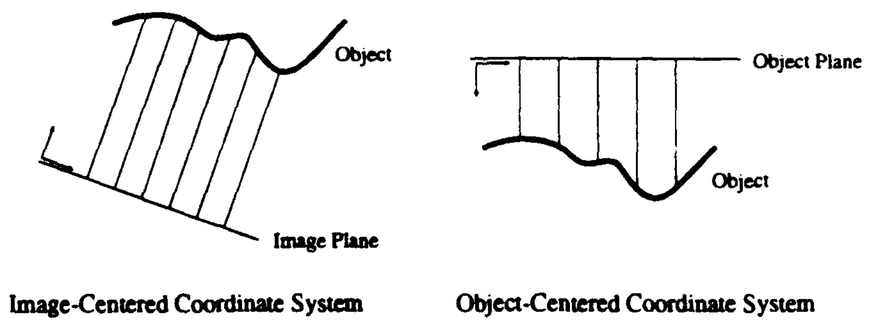

Object
Image Plane
Image-Centered Coordinate System
Object Plane
Object
Object-Centered Coordinate System

Figure 2-1: Coordinate system choices.

height value to each point in the domain (see Figure 2-1). Projecting points on the object to points in the image is straightforward using this representation for either orthographic or perspective projection. However, the projected points, won't in general map to the center of each pixel. To obtain values at each pixel then requires some type of interpolation.

In a stereo system, there is also the choice of whether the coordinate system should favor one image or the other. Since there is no apriori reason to believe that one image has better information than the other image, it is best to choose a neutral coordinate system rather than risk biasing the result toward one or the other image.

2.2 Image Generation Process

The photo-topography problem is basically an inverse problem. We seek to determine the topography of the object that created the images at hand. Those images are based on the object (a forward problem). That is, the interaction of light with an object, as seen by the viewer creates the image. The physics behind this process are detailed in the sections that follow.

The image generation process can be neatly broken into four stages (see also Figure 2-2),

1. light falls upon an object (object irradiance)
2. the light interacts with the object and is re-emitted or reflected (object radiance)
3. the light then travels to the viewer where it is projected onto the image plane. (image projection and image irradiance)
4. the light is absorbed by the material of the image plane and converted into some signal that can be sensed (image transduction).

2.2.1 Object irradiance

The light that falls on a particular patch of an object depends on the properties of the light sources that are visible (i.e., unobstructed) from that patch. Among possible light sources are point sources, distributed sources, and other surface patches (e.g., interflection). At each point, ξ, on the surface, and for each direction, $\hat{\mathbf{s}}$, the irradiance distribution function, $E(\xi;\hat{\mathbf{s}})$ [radiance/solid angle] captures this information. For instance, the distribution function for a single point source, ignoring interflection, is given by

$$E(\xi;\hat{\mathbf{s}}) = \lambda\hat{\delta}(\hat{\mathbf{s}},\hat{\mathbf{s}}_0) \tag{2.2}$$

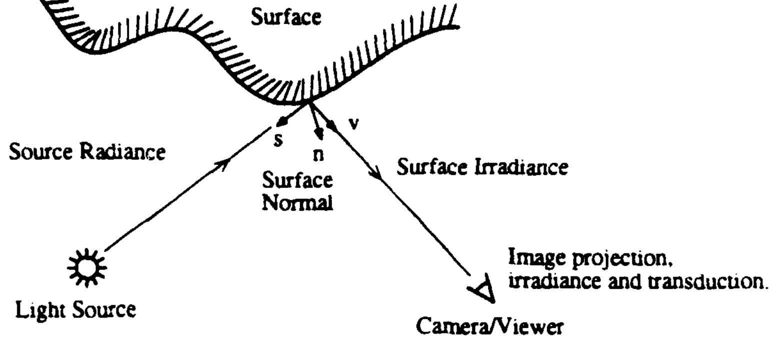


Figure 2-2: Image generation process steps.

where λ is the light intensity, $\hat{\mathbf{s}}_0$ is the unit vector in the light source direction, and $\hat{\delta}$ is a direction vector version of the Dirac delta[2]

$$\begin{aligned} &\hat{\delta}(\hat{\mathbf{t}}, \hat{\mathbf{t}}_0) = 0, \text{when } \hat{\mathbf{t}} \neq \hat{\mathbf{t}}_0, \\ &\iint_S f(\hat{\mathbf{t}})\hat{\delta}(\hat{\mathbf{t}}, \hat{\mathbf{t}}_0)\, d\hat{\mathbf{t}} = f(\hat{\mathbf{t}}_0), \end{aligned} \tag{2.3}$$

where the integral is over the surface of the sphere S.

2.2.2 Object Radiance

The light that is emitted by a surface depends both on the light that impinges on the surface and the surface reflectance properties. The irradiance distribution function, $E(\xi; \hat{\mathbf{s}})$ captures all the needed information about the light source, while the surface reflectance properties can be described using the *Bidirectional Reflectance Distribution Function* (BRDF). The BRDF is an intrinsic function of the surface and doesn't depend on surface irradiance. At the point ξ, the BRDF $f(\xi; \hat{\mathbf{n}}; \hat{\mathbf{v}}; \hat{\mathbf{s}})$ relates the brightness of the surface patch with normal $\hat{\mathbf{n}}$ illuminated from the direction $\hat{\mathbf{s}}$ and as seen from the direction $\hat{\mathbf{v}}$. Using these two distribution functions, the surface radiance [power/solid angle/area] in the direction $\hat{\mathbf{v}}$ can then be defined as

$$L(\xi; \hat{\mathbf{n}}; \hat{\mathbf{v}}) = \iint_{H(\hat{\mathbf{n}})} f(\xi; \hat{\mathbf{n}}; \hat{\mathbf{v}}; \hat{\mathbf{s}}) E(\xi; \hat{\mathbf{s}})(\hat{\mathbf{s}} \cdot \hat{\mathbf{n}}) d\omega(\hat{\mathbf{s}}) \tag{2.4}$$

[2]This delta function is defined on the surface of the sphere. One way to define it is to represent the direction $\hat{\mathbf{t}}$ in spherical coordinates, $\hat{\mathbf{t}} = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)^T$, so that we require $\int_{-\pi/2}^{\pi/2}\int_0^{2\pi} \hat{\delta}(\hat{\mathbf{t}}(\varphi, \theta), \hat{\mathbf{t}}_0(\varphi_0, \theta_0)) \cos\theta\, d\varphi\, d\theta = 1$, hence,

$$\hat{\delta}(\hat{\mathbf{t}}(\varphi, \theta), \hat{\mathbf{t}}_0(\varphi_0, \theta_0)) = \delta(\varphi - \varphi_0)\delta(\theta - \theta_0)/cos(\theta_0),$$

where $\delta(.)$ is the normal scalar Dirac delta.

where $H(\hat{\mathbf{n}})$ is the hemisphere of possible light source directions for the patch at surface point ξ with normal $\hat{\mathbf{n}}$, and $d\omega(\hat{\mathbf{s}})$ is the solid angle subtended in the direction $\hat{\mathbf{s}}$. Representing $\hat{\mathbf{s}}$ in spherical coordinates this expression becomes

$$L(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = \int_{-\pi}^{\pi}\int_{0}^{\pi/2} f(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}};\hat{\mathbf{s}})E(\xi;\hat{\mathbf{s}})\sin\theta\cos\theta\, d\theta\, d\phi. \tag{2.5}$$

The two most common reflectance models are the Lambertian model for matte surfaces, and the specular model for shiny metallic surfaces. In the Lambertian case, the BRDF is independent of the light source direction, normal direction and viewing direction,

$$f(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}};\hat{\mathbf{s}}) = \frac{1}{\pi}\rho(\xi) \tag{2.6}$$

where $\rho(\xi)$ is the surface albedo, or the fraction of light re-emitted by the surface. Evaluating Equation 2.5 for a Lambertian surface illuminated by a single point source at infinity, the surface radiance is found to be

$$L(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = \frac{\lambda\rho(\xi)}{\pi}(\hat{\mathbf{s}}_0\cdot\hat{\mathbf{n}}). \tag{2.7}$$

In the specular case, all the light from the direction $\hat{\mathbf{s}}_0$ is reflected into the direction $2(\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}_0)\hat{\mathbf{n}} - \hat{\mathbf{s}}_0$ so the BRDF is

$$f(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}};\hat{\mathbf{s}}) = \rho(\xi)\hat{\delta}(\hat{\mathbf{v}}, 2(\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}_0)\hat{\mathbf{n}} - \hat{\mathbf{s}}_0). \tag{2.8}$$

For a single point source at infinity, the surface radiance for a such a surface is found to be

$$L(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = \lambda\rho(\xi)\hat{\delta}(\hat{\mathbf{v}}, 2(\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}_0)\hat{\mathbf{n}} - \hat{\mathbf{s}}_0) \tag{2.9}$$

In general, the albedo of a Lambertian surface is different from the albedo of a specular surface.

See [Wilson *et al.*, 1985], [McEwen, 1985], and [Davis and McEwen, 1984] for a sampling of the types of radiance functions used within the photoclinometry field.

2.2.3 Reflectance Map

The preceding representation of the surface radiance is a local representation in that the radiance is defined based on local surface orientation, viewer direction and light source direction. Many vision researchers have found it convenient to use a representation of the surface radiance based on a global coordinate system. This alternate representation is called the Reflectance function. Given known surface properties and a known light source distribution, the reflectance function, $R(\xi, \hat{\mathbf{n}}; \hat{\mathbf{v}})$, is defined using the corresponding surface radiance via a change in coordinates,

$$R(\xi_G;\hat{\mathbf{n}}_G;\hat{\mathbf{v}}_G) = L(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) \tag{2.10}$$

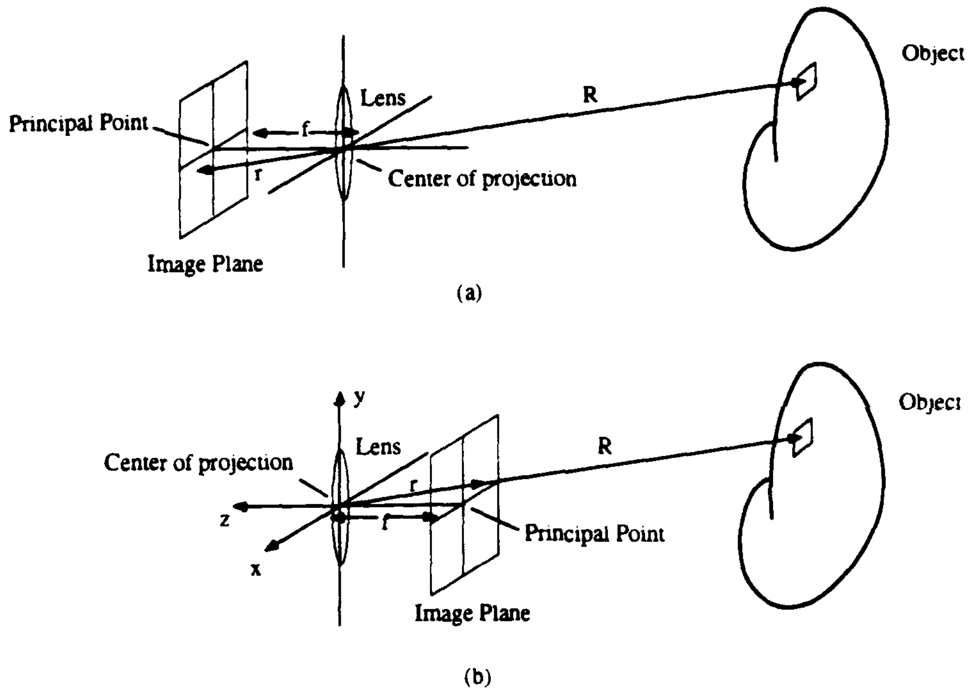


Figure 2-3: Two Perspective projection geometries.

where ξ_G, $\hat{\mathbf{n}}_G$, and $\hat{\mathbf{v}}_G$ are the surface position, normal vector, and viewing direction in global coordinates.

2.2.4 Image Projection

Points in the image plane are related to points on the object via perspective projection. Figure 2-3 shows two perspective projection geometries. Figure 2-3(a) shows the projection geometry for a camera where the image plane is behind the lens. The projection of the center of the lens into the image plane is called the principal point.[3] Choosing the origin of the image coordinate system to be at the principal point leads to the simple equations presented in this section.[4] Note that objects are projected onto the image plane in an upside down orientation.

An equivalent geometry is shown in Figure 2-3(b). The only difference between the camera geometry shown in Figure 2-3(a) and this geometry is that the image plane has been moved to in front of the lens. While this geometry is not physically possible.

[3]The center of projection is at the back nodal point for thick lenses.

[4]Determining the position of the principal point for a given camera is part of the classic *interior orientation* problem (see [Horn, 1986]).

it produces exactly the same image (as can be seen by similar triangles) except that the image is no longer upside down. Mathematically, the difference between these two geometries is just a sign change in the equations. For instance, the image position **r** of the object point **R** in Figure 2-3(a) is

$$\frac{\mathbf{r}}{\mathbf{r}\cdot\hat{\mathbf{z}}} = -\frac{\mathbf{R}}{\mathbf{R}\cdot\hat{\mathbf{z}}}, \tag{2.11}$$

while it is

$$\frac{\mathbf{r}}{\mathbf{r}\cdot\hat{\mathbf{z}}} = \frac{\mathbf{R}}{\mathbf{R}\cdot\hat{\mathbf{z}}} \tag{2.12}$$

for the geometry in Figure 2-3(b). These are *Perspective Projection Equations.* In both expressions, $\hat{\mathbf{z}}$ is a unit vector in the direction of the negative optical axis. In the equations that follow, the geometry of Figure 2-3(b) will be used to avoid having to keep track of the minus sign in the perspective projection equation.

The equations above can be simplified if a special coordinate system is used, namely a coordinate system with origin at the principal point, and orientation such that the x and y axes are aligned with the image coordinates and with the negative z axis along the optical axis of the lens. This coordinate system is evident in the labels assigned to the axes in Figure 2-3(b). For this coordinate system, $\mathbf{r}\cdot\hat{\mathbf{z}} = f$ (a negative number). Expanded, these equations are

$$\begin{pmatrix} x/f \\ y/f \\ f/f \end{pmatrix} = \begin{pmatrix} X/Z \\ Y/Z \\ Z/Z \end{pmatrix} \tag{2.13}$$

when $\mathbf{r} = (x, y, f)^T$ and $\mathbf{R} = (X, Y, Z)^T$.

2.2.5 Orthographic Projection

The projection equations can be further simplified when the depth range of the object is small compared to the distance of the object from the camera. The resulting approximation is called *Orthographic Projection.* Consider the first order Taylor's series expansion of the perspective projection equation about a nominal depth Z_0.

$$\frac{\mathbf{r}}{f} = \frac{\mathbf{R}}{Z_0} + \frac{\mathbf{R}}{Z_0^2}(Z - Z_0) + O((Z - Z_0)^2) \tag{2.14}$$

where $\mathbf{R} = (X, Y, Z_0)^T$. When $(Z - Z_0) \ll Z_0$, the first order term can be neglected resulting in the orthographic approximation,

$$\frac{\mathbf{r}}{f} = \frac{\mathbf{R}}{Z_0}. \tag{2.15}$$

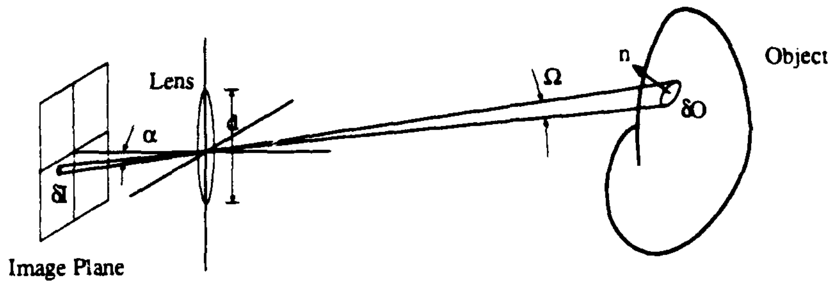


Figure 2-4: Image irradiance.

Orthographic projection is convenient for computer vision problems since the mapping is linear. In fact, most shape-from-shading algorithms assume orthographic projection. Unfortunately, the full perspective projection must be used for phototopography to avoid throwing out the very information we wish to estimate (i.e., the stereo depth information).

2.2.6 Image Irradiance

The mapping between object points and image points is only half of the story for image generation. We also need to know how the brightness of the image is affected by the lens. Figure 2-4 shows that an object patch of area δO projects to an image patch of area δI. Assuming a perfect lens, the image irradiance [power/area] from this patch is

$$E(\mathbf{r}) = L(\xi(\mathbf{r}); \hat{\mathbf{n}}; \hat{\mathbf{v}}(\mathbf{r}))\frac{\pi}{4}\left(\frac{d}{f}\right)^2 \cos^4\alpha \qquad (2.16)$$

where $L(\xi(\mathbf{r}); \hat{\mathbf{n}}; \hat{\mathbf{v}}(\mathbf{r}))$ is the radiance of the corresponding object patch, d is the diameter of the lens, and α is the off-axis angle of the projecting ray. In this equation, $\hat{\mathbf{v}}$ and ξ are functions of $\mathbf{r}$ via the projection from image points $\mathbf{r}$ to object points. Equivalently, this expression can be restated using the reflectance function rather than the radiance function.

$$E(\mathbf{r}) = R(\xi(\mathbf{r}); \hat{\mathbf{n}}; \hat{\mathbf{v}}(\mathbf{r}))\frac{\pi}{4}\left(\frac{d}{f}\right)^2 \cos^4\alpha \qquad (2.17)$$

2.2.7 Image Transduction

The final stage in producing an image is the conversion of light into a signal that can be used, namely digital form. Whether the digital images are scanned photographs or are obtained directly via a digital camera, they will contain distortion (either spatially or in color). The effect of this distortion can be removed via calibration so that the measured image irradiance can be related to the object radiance in a

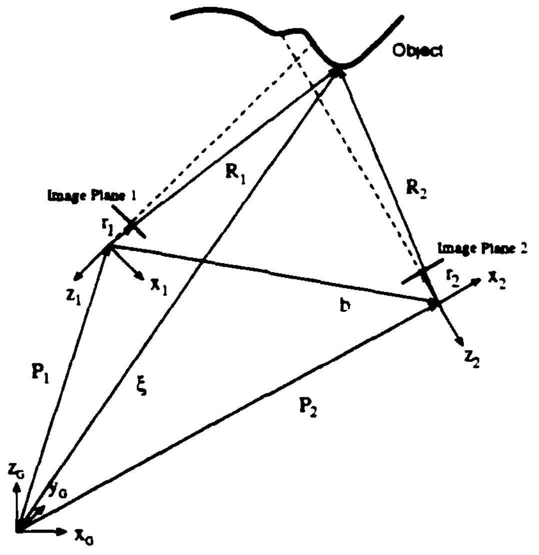


Figure 2-5: Stereo geometry.

straightforward way. With perfect calibration, the image irradiance equation can be put in the especially simple form,

$$E(\mathbf{r}) = R(\xi(\mathbf{r}); \hat{\mathbf{n}}; \hat{\mathbf{v}}(\mathbf{r})) \tag{2.18}$$

where the $\cos^4 \alpha$ and constant scale factors have been removed as part of the calibration. Equation 2.18 is referred to as the *Image Irradiance Equation*.

2.3 Stereo

The images in a photo-topography problem are taken from two different vantage points (see Figure 2-5). If the relative position of the cameras is known, and it is known which pixels in the left image correspond to which pixels in the right image, then it is possible to determine the depth directly for the surface points that map to those pixels. This is the basis of binocular stereo. Of course, the hard part is determining the correspondence between pixels.

For normal stereo situations, the cameras are close together and both pictures are taken simultaneously.[5] The stereo images that result look very similar, mostly differing in the shift of objects in each image caused by perspective projection. The difference in the shift of an object point in the left image and the right image is called the *disparity*. The photo-topography images are taken with cameras that are often far

[5]The stereo images of aerial photography are taken nearly simultaneously.

apart and at different times. In this case, it is possible (because of differing lighting) for the two images to look very different. This makes the correspondence problem even harder.

To see how the depth can be determined directly from the disparity consider Figure 2-5. Points in each image map to rays in 3-space. If corresponding points in each image map to rays that intersect, the depth can be determined using simple geometry. If the rays don't intersect, we can use geometry to determine the depth of the point that is halfway between the rays at their closest approach.

First, find the relationship between the two camera coordinate systems. Suppose we know the position of the principal point of each camera in some global coordinate system, $\mathbf{P}_1$ and $\mathbf{P}_2$, and we also know the rotational transformation matrices from each local camera coordinate system to the global coordinate system, T_1 and T_2, then the coordinates of the point, ξ, in the two camera coordinate systems is

$$\begin{aligned} \mathbf{R}_1 &= T_1^{-1}(\xi - \mathbf{P}_1) \\ \mathbf{R}_2 &= T_2^{-1}(\xi - \mathbf{P}_2). \end{aligned} \tag{2.19}$$

Equations 2.19 are the *Stereo Constraint Equations* which relate the position of points in the global coordinate system to points in each of the local camera coordinate systems.

By removing ξ from the above equations and defining $\mathbf{b} = \mathbf{P}_2 - \mathbf{P}_1$, the relationship between a point in Camera Coordinate System 1 and a point in Camera Coordinate System 2 is found to be

$$\mathbf{R}_1 = T_1^{-1}\mathbf{b} + T_1^{-1}T_2\mathbf{R}_2. \tag{2.20}$$

Now, determine the relationship between disparity and depth. Suppose we are given image points, $\mathbf{r}_1 = (x_1, y_1, f)^T$ and $\mathbf{r}_2 = (x_2, y_2, f)^T$ (one in each image), that correspond to the same surface point, then the best estimate for the surface position can be found by finding the point on each ray (along $\mathbf{r}_1$ and $\mathbf{r}_2$) where the distance between the rays is minimized. That is, the problem

$$\min_{s,t} \|-s\mathbf{r}_1 + \mathbf{b} + t\mathbf{r}_2\|^2 \tag{2.21}$$

must be solved where s and t are scalar parameters. For now assume all the vectors are given on the same basis.

By differentiating the above equation with respect to s and t, setting the resulting equations to zero, and solving, it is found that the minimum occurs when

$$\begin{aligned} s &= \frac{(\mathbf{r}_2 \cdot \mathbf{r}_2)(\mathbf{b} \cdot \mathbf{r}_1) - (\mathbf{r}_1 \cdot \mathbf{r}_2)(\mathbf{b} \cdot \mathbf{r}_2)}{(\mathbf{r}_1 \cdot \mathbf{r}_1)(\mathbf{r}_2 \cdot \mathbf{r}_2) - (\mathbf{r}_1 \cdot \mathbf{r}_2)^2} = \frac{(\mathbf{r}_2 \times \mathbf{b}) \cdot (\mathbf{r}_2 \times \mathbf{r}_1)}{\|\mathbf{r}_2 \times \mathbf{r}_1\|^2}, \\ t &= \frac{(\mathbf{r}_1 \cdot \mathbf{r}_2)(\mathbf{b} \cdot \mathbf{r}_1) - (\mathbf{r}_1 \cdot \mathbf{r}_1)(\mathbf{b} \cdot \mathbf{r}_2)}{(\mathbf{r}_1 \cdot \mathbf{r}_1)(\mathbf{r}_2 \cdot \mathbf{r}_2) - (\mathbf{r}_1 \cdot \mathbf{r}_2)^2} = \frac{(\mathbf{r}_1 \times \mathbf{b}) \cdot (\mathbf{r}_2 \times \mathbf{r}_1)}{\|\mathbf{r}_2 \times \mathbf{r}_1\|^2}. \end{aligned} \tag{2.22}$$

If $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{b}$ are coplanar then the above set of equations is just a fancy way of writing the Law of Sines.

The global position of the point halfway between the two rays at the closest approach is then

$$\xi \approx \mathbf{P}_1 + s\mathbf{r}_1 + \frac{1}{2}(-s\mathbf{r}_1 + \mathbf{b} + t\mathbf{r}_2), \tag{2.23}$$

$$= \mathbf{P}_1 + \frac{\mathbf{b}}{2} + \frac{1}{2}\frac{(T_2\mathbf{r}_2 \times \mathbf{b})\cdot(T_2\mathbf{r}_2 \times T_1\mathbf{r}_1)}{\|T_2\mathbf{r}_2 \times T_1\mathbf{r}_1\|^2}T_1\mathbf{r}_1 + \frac{1}{2}\frac{(T_1\mathbf{r}_1 \times \mathbf{b})\cdot(T_2\mathbf{r}_2 \times T_1\mathbf{r}_1)}{\|T_2\mathbf{r}_2 \times T_1\mathbf{r}_1\|^2}T_2\mathbf{r}_2 \tag{2.24}$$

where Equation 2.24 has been written for $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{b}$ given in global coordinates, and $\mathbf{r}_1$, $\mathbf{r}_2$ given in the appropriate local camera coordinate system.

The equations simplify greatly if the $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{b}$ are coplanar (i.e., if $(\mathbf{r}_2 \times \mathbf{r}_1)\cdot\mathbf{b} = 0$), the cameras are aligned so that the optical axes are in the direction $-\hat{\mathbf{z}}$, and $\hat{\mathbf{x}}$ is along $\mathbf{b}$ ($\hat{\mathbf{y}}$ is chosen to complete the right-handed coordinate system). This is exactly the stereo geometry that is assumed to exist for most binocular stereo algorithms. With these restrictions, the equation above becomes

$$\xi = \mathbf{P}_1 + \frac{\mathbf{b}}{2} + \frac{b(\mathbf{r}_1 + \mathbf{r}_2)}{2(x_1 - x_2)}. \tag{2.25}$$

where $\mathbf{r}_1 = (x_1, y_1, f)^T$, $\mathbf{r}_2 = (x_2, y_2, f)^T$, and $\mathbf{b} = (b, 0, 0)^T$. The quantity $(x_2 - x_1)$ in the above equation is the disparity mentioned earlier. In this form, the depth z is found to be

$$z = \xi \cdot \hat{\mathbf{z}} = \mathbf{P}_1 \cdot \hat{\mathbf{z}} + \frac{b}{2} + \frac{bf}{(x_1 - x_2)}. \tag{2.26}$$

Note that the disparity can be mapped directly into depth only in this special situation. For more general situations, Equation 2.24 must be used.

2.4 Photo-topography Problem Formulaton

We now have enough background to formulate the photo-topography problem. The problem to be solved is:

> Given two images of an area on the planet's surface, taken at two different times from two different positions, determine the topography of that section of the planet's surface.

The solution is constrained by geometry and the image generation process. Specifically each image is constrained by the perspective projection equation (Equation 2.12), the image irradiance equation (Equation 2.18) and the stereo constraint equations (Equations 2.19).

Combining these equations we find that the photo-topography problem is constrained such that

$$E^{(1)}(\mathbf{r}_1) = R^{(1)}(\xi; \hat{\mathbf{n}}; -T_1\mathbf{r}_1)$$
$$E^{(2)}(\mathbf{r}_2) = R^{(2)}(\xi; \hat{\mathbf{n}}; -T_2\mathbf{r}_2) \tag{2.27}$$

where

$$\mathbf{r}_1 = \frac{fT_1^{-1}(\xi - \mathbf{P}_1)}{T_1^{-1}(\xi - \mathbf{P}_1) \cdot T_1\hat{\mathbf{z}}_1},$$
$$\mathbf{r}_2 = \frac{fT_2^{-1}(\xi - \mathbf{P}_2)}{T_2^{-1}(\mathbf{R}_G - \mathbf{P}_2) \cdot T_1\hat{\mathbf{z}}_2}, \tag{2.28}$$

are the local image plane position vectors (one in each camera coordinate system). $E^{(1)}$ and $E^{(2)}$ are the image brightness measured in the first and second cameras respectively, and $R^{(1)}$ and $R^{(2)}$ are the reflectance maps based on the first and second light source positions. As before, ξ, is the surface position in global coordinates.

Equations 2.27 and 2.28 are the *General Photo-topography Constraint equations.* They state that in the absence of noise, the photo-topography images $E^{(i)}$ are created by the interaction of light with the underlying surface $z(\xi)$ via the reflectance maps $R^{(i)}$ and as seen by the two cameras. Note that these equations can be used to generate a set of photo-topography images given the surface description, $z(\xi)$. In contrast, determining the surface topography given the images, camera geometry and reflectance properties, is an inverse problem.

It's instructive to review the assumptions behind these equations. The perspective projection equations assume perfect lenses and perfect knowledge of the camera principal points and optical axes. The surface radiance equation assumes we have perfect knowledge of the surface reflectance properties, light source directions, and all surface points are visible from both cameras (i.e., there are no self occlusions). The simplified form of the image irradiance equation assumes we either have a perfect sensor or we can perfectly calibrate the sensor to remove any abnormalities from the sensor/lens combination. The stereo equations assume we know the relative position and orientation of the two cameras perfectly. The only assumption that is truly artificial is the assumption of perfect knowledge of the reflectance maps.[6] With more careful measurements and more expensive equipment it is possible to approach perfect knowledge of the other assumptions. The assumption that all surface points be visible merely restricts the bumpiness of the surface that this research is applicable to.

[6]Especially since the effects of non-uniform albedo and interflection are buried in the reflectance maps.

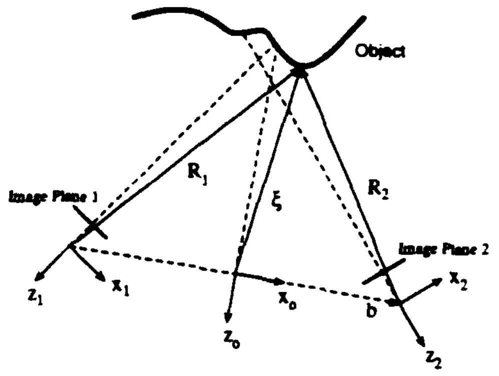


Figure 2-6: Introduced global coordinate system.

2.5 Simplifications

The equations in the last section dealt with the general case (subject to the assumptions mentioned in that section). There are several simplifications that make the equations easier to solve.

2.5.1 Special global coordinate system.

So far all the equations have been written for any global coordinate system. I would now like to restrict the equations to a particular global coordinate system, namely the coordinate system shown in Figure 2-6. This coordinate system is defined as follows

1. Place the origin of the global coordinate system half way between the principal points of the two cameras. (That is, place the origin at the point $\mathbf{P}_1 + \mathbf{b}/2$.)
2. Choose the $\hat{\mathbf{x}}_0$ direction along the line connecting the two cameras,
$$\hat{\mathbf{x}}_0 = \mathbf{b}/||\mathbf{b}||. \tag{2.29}$$
3. Choose $\hat{\mathbf{z}}_0$ as the average optical axis direction of the two cameras projected into the plane perpendicular to $\mathbf{x}_0$. (That is,
$$\hat{\mathbf{z}}_0 = \frac{(\hat{\mathbf{z}}_1 - (\hat{\mathbf{z}}_1 \cdot \hat{\mathbf{x}}_0)\hat{\mathbf{x}}_0) + (\hat{\mathbf{z}}_2 - (\hat{\mathbf{z}}_2 \cdot \hat{\mathbf{x}}_0)\hat{\mathbf{x}}_0)}{||(\hat{\mathbf{z}}_1 - (\hat{\mathbf{z}}_1 \cdot \hat{\mathbf{x}}_0)\hat{\mathbf{x}}_0) + (\hat{\mathbf{z}}_2 - (\hat{\mathbf{z}}_2 \cdot \hat{\mathbf{x}}_0)\hat{\mathbf{x}}_0)||}. \tag{2.30}$$
4. Choose $\hat{\mathbf{y}}_0$ in the direction of $\hat{\mathbf{z}}_0 \times \hat{\mathbf{x}}_0$ in order to create a right handed coordinate system.
5. Also set up a virtual image plane with $f = 1$.

In this coordinate system, $\mathbf{P}_1 = -\mathbf{b}/2$, $\mathbf{P}_2 = \mathbf{b}/2$, and $\mathbf{b} = (b, 0, 0)^T$.

2.5.2 Removing the view direction dependence.

A common simplification for computer vision is the assumption of a Lambertian reflectance map. Since a Lambertian surface reflects light equally in all directions, we see from Equation 2.7 that the radiance function is not dependent on the viewing direction. Thus the dependence on $\hat{\mathbf{v}}$ can be removed from all the equations.[7]

Note that the viewing direction can also be removed from the equations when the field of view is small. In this case, the viewing direction is approximately the same for all points on the surface and its effect can be subsumed into the reflectance map. Doing this would, of course, introduce an error into the calculations. This error would be small for photo-topography since the cameras are so far away from the surface. The large viewing distance requires the use of a telephoto lens which has a small field of view.

2.5.3 Constant albedo.

Thus far the equations have included terms that denote position on the surface ξ. The main reason for this dependence is to take into account varying albedo, varying reflectance properties, or both. To simplify the situation we could assume that the reflectance properties, albedo, or both are constant across the surface. Assuming the reflectance properties, but not the albedo, are constant across the surface results in a reflectance function that is separated,

$$R(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = \rho(\xi)\bar{R}(\hat{\mathbf{n}};\hat{\mathbf{v}}) \tag{2.31}$$

where $\bar{R}(\hat{\mathbf{n}};\hat{\mathbf{v}})$ is the reflectance function for a surface with uniform albedo, no interflection, and no self occlusion. As for R, any light source effects are included in $\bar{R}$.

When both the reflectance and albedo are constant, the dependence of the reflectance map on surface position can be removed,

$$R(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = R(\hat{\mathbf{n}};\hat{\mathbf{v}}). \tag{2.32}$$

Combined with either Lambertian reflectance[8] or when the field of view is small, the dependence on $\hat{\mathbf{v}}$ can be dropped also,

$$R(\xi;\hat{\mathbf{n}};\hat{\mathbf{v}}) = R(\hat{\mathbf{n}}). \tag{2.33}$$

This is the representation of the reflectance function that is seen most often in the vision literature. The simplification restricts the applicability of that research so

[7]This is true for any reflectance function that is view independent, not just Lambertian reflectance.

[8]Or any other view independent reflectance function.

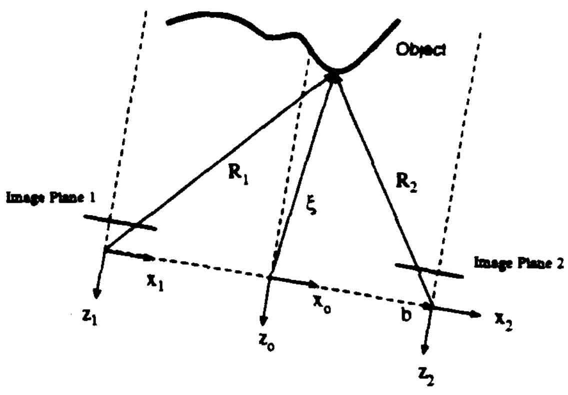


Figure 2-7: Coordinate systems with aligned optical axes.

that only uniformly colored surface patches have the possibility of being estimated correctly. When algorithms based on this simplification are applied to images that violate these simplifications, we would expect errors at the transition between different colored parts of the surface, within differently colored areas, or both. Trying to estimate both the surface topography and surface albedo substantially increases the number of unknown variables (possibly by a factor of 2) and significantly slows down the convergence (see Section 8.1).

2.5.4 Aligned cameras.

The final simplification that can be made is to align the cameras so that their optical axes are parallel (which will also be parallel to the global coordinate system's $\hat{\mathbf{z}}_0$ axis by construction). This coordinate system is shown in Figure 2-7. When the cameras are aligned, the rotational transformations T_1 and T_2 are identity transformations which simplifies the stereo equations (Equations 2.19) considerably to

$$\begin{aligned} \mathbf{R}_1 &= \xi - \mathbf{P}_1, \\ \mathbf{R}_2 &= \xi - \mathbf{P}_2. \end{aligned} \tag{2.34}$$

While this situation is very unrealistic for the photo-topography problem, any set of images can be re-projected into this coordinate system (see Section 8.5). Hence this simplification does not seriously restrict the applicability of the research.

2.5.5 The simplified equations.

The rest of the thesis is based on equations that take into account all of the simplifications described. In particular, the simplifications in the following list are made:

- A special global coordinate system that is halfway between the two camera positions is used.
- All surface points are assumed to be visible from the two cameras.
- The reflectance properties of the surface are assumed to be constant with no interflection and no mutual occlusion.
- The radiance properties of the surface are assumed to be Lambertian, allowing the view direction dependence to be dropped from the reflectance equations.
- The surface is assumed to have constant albedo allowing the position dependence to be dropped from the equations.
- The camera optical axes are assumed to be aligned with each other allowing the rotational transforms to be dropped from the stereo constraint equations.

Unless noted, the rest of the thesis assumes that all these simplifications hold. I will specifically point out results that apply to the more general case. Taking into account all of these simplifications we find that the simplified photo-topography problem is constrained such that

$$\begin{aligned} E^{(1)}(\mathbf{r}_1) &= R^{(1)}(\hat{\mathbf{n}}) \\ E^{(2)}(\mathbf{r}_2) &= R^{(2)}(\hat{\mathbf{n}}) \end{aligned} \tag{2.35}$$

where

$$\begin{aligned} \mathbf{r}_1 &= \frac{(\xi + \mathbf{b}/2)f}{(\xi + \mathbf{b}/2)\cdot\hat{\mathbf{z}}_1} = \frac{\xi + \mathbf{b}/2}{\xi\cdot\hat{\mathbf{z}}_1}. \\ \mathbf{r}_2 &= \frac{(\xi - \mathbf{b}/2)f}{(\xi - \mathbf{b}/2)\cdot\hat{\mathbf{z}}_2} = \frac{\xi - \mathbf{b}/2}{\xi\cdot\hat{\mathbf{z}}_2}. \end{aligned} \tag{2.36}$$

If we define $z = \xi \cdot \hat{\mathbf{z}}_0$ and $\mathbf{r} = f\xi/z$, then the constraint equations can be written

$$\begin{aligned} E^{(1)}(\mathbf{r} + \frac{f\mathbf{b}}{2z}) &= R^{(1)}(\hat{\mathbf{n}}), \\ E^{(2)}(\mathbf{r} - \frac{f\mathbf{b}}{2z}) &= R^{(2)}(\hat{\mathbf{n}}). \end{aligned} \tag{2.37}$$

The definitions for z and $\mathbf{r}$ above introduces a peculiarity into the equations: the units for z and $\mathbf{r}$ are not the same. In particular, z, b, and ξ are in planet units (say miles), while $\mathbf{r}$ and f are in camera units (say millimeters).

I find it convenient in subsequent chapters to use a slightly different version of these last equations where the components of $\mathbf{r} = (x, y, f)$ are explicit and the normal vector $\hat{\mathbf{n}}$ is parameterized using gradient components p and q.

$$\hat{\mathbf{n}} = \frac{(-p, -q, 1)}{\sqrt{p^2 + q^2 + 1}} \tag{2.38}$$

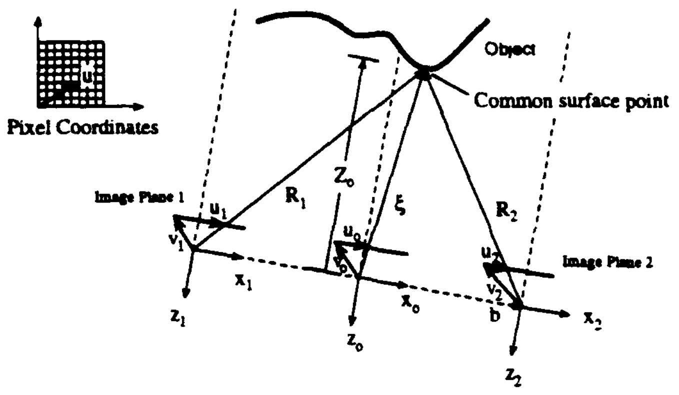


Figure 2-8: Camera calibration geometry.

where

$$\begin{aligned} p &= \frac{f z_x}{x z_x + z}, \\ q &= \frac{f z_y}{y z_y + z}. \end{aligned} \tag{2.39}$$

Equation 2.39 are referred to as the *Integrability Constraint Equations* since they relate the surface normal components to the partial derivatives of the surface height z. The equations can be used constraint the values of p and q that are consistent with an underlying surface.

Writing Equations 2.37 in terms of (x, y) and p and q produces the *Photo-topography Constraint Equations*

$$\begin{aligned} E^{(1)}(x + \frac{fb}{2z}, y) &= R^{(1)}(p, q), \\ E^{(2)}(x - \frac{fb}{2z}, y) &= R^{(2)}(p, q). \end{aligned} \tag{2.40}$$

I will be working with these equations in the chapters that follow.

2.6 Camera Calibration

In order to relate positions in the image to direction vectors in 3-space, the origin of the camera coordinate system must be known. Finding this origin is part of the *interior orientation* problem of classical optics. As mentioned in Section 2.2.4, this origin is the principal point in the image plane. In the special coordinate system of Figure 2-8, the position of the image plane origin with respect to each camera coordinate system origin can be specified by a vector $\mathbf{v} = (v_x, v_y, f)^T$. These vectors specify the offset (in pixel coordinates) of the image plane for each camera. Suppose $\mathbf{u}_0$ is the pixel position of an object point in the global image and $\mathbf{v}_0$ is the position of the origin of the global image plane in global coordinates, then given the pixel position of the projection of this this same object point in the camera images $\mathbf{u}_1$ and $\mathbf{u}_2$, the offset of each image plane is

$$\mathbf{v}_1 = \mathbf{v}_0 + \mathbf{u}_0 - \mathbf{u}_1 + \frac{fb}{2Z_0}, \tag{2.41}$$

$$\mathbf{v}_2 = \mathbf{v}_0 + \mathbf{u}_0 - \mathbf{u}_2 - \frac{fb}{2Z_0}. \tag{2.42}$$

The values of $\mathbf{v}_1$ and $\mathbf{v}_2$ can be quite large in the aligned coordinate system indicating that the images must be shifted far away from the camera coordinate system origin.[9] While this is not possible physically, it is a consequence of re-projecting real images into the aligned coordinate system.[10]

[9]This is especially true with the field of view is small.

[10]See Section 8.5 for how to do this.

Chapter 3

Fusion Strategy

This chapter presents my general solution strategy to the photo-topography problem. The idea is to closely couple the solution of shape-from-shading and stereo so that each can help the other. The important thing is to take into account the strengths of each method and use each method to cover the weaknesses of the others.

The first step in solving the photo-topography problem is to analyze the information available as part of the problem. The photo-topography problem has two information sources: 1) the gray levels in the each image are an indication of the surface orientation with respect to the light source and 2) assuming corresponding pixels in each image can be matched up, the stereo information can be used to recover the shape. Of the two, the gray level information is more directly accessible since no correspondence must be found. Finding the correspondence between pixels is a classical problem for stereo algorithms and is very hard.

The photo-topography images also offer one other source of information. Since the images are typically taken with two different light source positions, the gray levels of corresponding pixels constrain the set of possible surface orientations as they would for photometric stereo [Woodham, 1980]. For Lambertian reflectance, this set contains at most two orientations. Since extracting this information relies on pixel correspondence, it has the same limitations as the stereo information.

When designing a fusion algorithm it is also important to analyze whether the information sources are independent or not. By independent I mean, 'Is it possible to differentiate between the two information sources'? Or said another way, 'Given a combined signal containing two or more information sources is it possible to estimate the relative contributions of each source'? Or said a third way, 'Given a combined signal are the information components observable'?

Doing this analysis on the photo-topography images we find that the information sources are indeed independent. The shading information is strongest when the shading is smooth, while the stereo information is strongest near surface discontinuities (which contribute to shading discontinuities) and when the cameras are widely separated. The photometric stereo information is strongest when the light source

	Shading	Stereo	Lighting
Shape from Shading	Shape from shading	Surface constraint	Surface orientation constraint
Binocular Stereo	Correspondence constraint	Binocular stereo	Correspondence constraint
Photometric Stereo	Correspondence constraint	Correspondence constraint	Photometric stereo

Table 3-1: Photo-topography problem information sources

positions are widely separated.

Given the previous analysis we would expect that the best estimates of the surface would be found when the images contain large regions of smooth gray level changes intermixed with areas of rapidly changing gray level, the cameras are well separated, and the light source positions are separated.

There are several possible fusion paradigms as I discussed in Chapter 1. The most promising fusion paradigms take into account the most amount of information. The modularized approaches (where existing vision algorithms are applied to the problem individually and where these individual estimates are then combined afterwards) exploit some of the information but not all of it. In particular, if you think of these problems in terms of a table or matrix as in Table 3-1, the modularized approaches exploit the information along the diagonal but do not take into account the off-diagonal coupling between methods. On the other hand, I use a close coupled approach based on variational calculus that exploits all the coupling inherent in the problem. This approach is not a panacea, however, as we will see later. It is still possible to choose cost functions that don't encourage cooperation among the various information sources. However, by understanding the couplings inherent in the problem a suitable cost function can usually be found.

3.1 Variable representations

The most important decision when developing a fusion algorithm is the variable representation. Choose the right representation and everything will work together smoothly. Choose the wrong representation and the algorithm may be hindered by slow convergence, local minima, discontinuities, etc. Aside from this performance effect, how can a good representation by recognized? I can't offer a general solution, but I can offer some rules of thumb.

- A good representation captures all the necessary information in the problem with no redundancy. For example it is better to use a parameterized family of functions rather than a digitized version of the same function if the number of parameters would be much less than the number of sampled points.

- A good representation can easily capture the constraints of the problem. For example, it is better to use a function that automatically matches the boundary conditions of a problem rather than to impose the constraint using penalty functions or Lagrange multipliers.
- A good representation usually has equations that are simple and easy to understand.
- A good representation is as close as possible to the problem. It is not removed by several integrations, differentiations, or other mappings that can introduce non-linearities, bias, drift, or additional unobservable parameters.
- A good representation allows for easy information exchange between the different information sources in the problem. Such a representation would directly express constraints from one information source so that the other sources will automatically take them into account.

The variable representation defines the space that will be searched. Usually, solution spaces that are lower dimensional, smoother, and more bowl-like will make finding the solution faster and easier.

I present four different variable representations for the photo-topography problem in Chapter 4. The four representations highlight the importance of choosing the right representation.

3.2 Cost functions

After the choice of variable representation, the choice of cost function is the next important. Like choosing the right variable representation, choosing the right cost function will affect the performance of the algorithm and the algorithm's robustness.

Building a closely coupled fusion algorithm, requires that all the governing equations, constraints, and desired outcomes be formulated into a single cost function, usually some sort of energy-based functional. Since everything is combined together to generate a single scalar value (the cost), the algorithm is free to perform trade-offs. For instance, the algorithm can trade-off accuracy of the result in order to satisfy another constraint (such as smoothness). The types of trade-offs that are allowed and/or favored can be controlled by changing the relative weights between terms in the cost function, or by totally reformulating the cost function (I show examples of both in Chapter 4).

Cost functions are relatively easy to formulate once a variable representation has been chosen. Simply write the governing equations in terms of these variables, and create the cost function directly from these equations. If necessary, sum, integrate or somehow combine the results from the constraints to form a scalar cost. It might be necessary to add some regularizing terms to the cost function in order to produce a well posed problem (see [Tikhonov and Arsenin, 1977] or [Hadamard, 1923]).

Many vision problems require some type of regularization to constrain the solution. Typically, smoothness is assumed for some variable (such as surface height) and is used to create a smoothness regularization term. Regularization terms are required when there are too many solutions to a given problem. The regularization terms can then be used to constrain the set of possible solutions to be physically reasonable. In all the photo-topography algorithms presented later in this chapter I have included a surface smoothness term. While the smoothness term is not strictly required (since shape-from-shading problems do have unique solutions),[1] the regularization term serves to speed up convergence by making the solution space more bowl-like.

Constraints can be added to a candidate cost function using the method of *Lagrange multipliers* or by adding a *penalty function*. The Lagrange multiplier method is used for ***hard constraints***, while the penalty functions are used for ***soft constraints***. Hard constraints are required to be met exactly by the solution, while soft constraints need not be met exactly. Hard constraints restrict the space of the feasible solutions, while soft constraints create bowl-like edges on the solution space. Both methods can be used to help convergence to a solution and to restrict the search space to more desirable solutions.

Different cost functions can be generated from the same set of equations and variable representation by changing the desired outcome. For instance, the cost function for determining the surface slopes from the photo-topography problem would be different from the cost function for finding the surface height.

3.3 Solution Techniques

A typical cost function for a vision problem is of the form

$$\min_u J = \iint L(u; u'; u''; \ldots)\, dx\, dy \tag{3.1}$$

where u are the optimization variables, and L is a possibly non-linear function of the optimization variables and its derivatives. The integral is taken over the domain of the optimization variables or the problem. The solution to this problem can be found by solving the associated *Euler-Lagrange* equations (see a variational calculus book such as [Courant and Hilbert, 1962] or Horn's Appendix [Horn, 1986] for more details).

The Euler-Lagrange equations for a problem such as the one above are typically coupled non-linear equations. Such equations are usually very difficult to solve ana-

[1]When the images contain singular points [Saxberg, 1989].

lytically but can sometimes be solved numerically by converting them into discrete equations. The conversion process involves substituting discrete approximations for any derivatives of the optimization variables. The optimization variables may have to be approximated by a discrete vector as well. The equations are then re-arranged to create iterative update equations of the form

$$u(k+1) = f(u(k), u(k-1), \ldots) \tag{3.2}$$

where $u(k)$ is the value of the optimization variables for the k-th iteration.

When u has many components and when the components are updated in sequence based on the most current estimate u, the resulting update scheme is called a *Gauss-Seidel* optimization. When all of the components of u are updated simultaneously based on a previous estimate for u, the resulting scheme is called a *Gauss-Jordan* optimization. Gauss-Seidel optimization schemes have higher convergence rates and are more robust, and are best implemented on a serial computer. Gauss-Jordan schemes, while they have lower convergence rates and are not as robust, can be implemented on parallel computers. Parallel computations can produce results faster.

Another way of solving the optimization problem posed above is by using direct optimization techniques. In this case the cost function, instead of the Euler-Lagrange equations, is discretized. Any integrations are approximated by sums and any derivatives are approximated by differences. The resulting cost function is of the form

$$\min_u J = \sum_x \sum_y f(u) \tag{3.3}$$

where $f(u)$ is a discrete approximation of $L(u; u'; u'; \ldots)$. Any of the wide range of optimization algorithms that are in the literature can then be applied to this problem. For vision problems, ***conjugate gradient*** optimization shows the most promise. The conjugate gradient scheme doesn't require the formation of the problem Hessian (a linear approximation to the second order derivative of the solution space at a given point), which for an optimization problem with N variables is an N-by-N matrix. The conjugate gradient scheme is important for vision problems since for a typical 256-by-256 image, the shape-from-shading problem would have 256^2 or 65536 optimization variables. The Hessian for this problem would have 256^4 or over 4 billion elements!

Conjugate gradient optimization requires that both the cost function and its gradient are computable. While an approximation to the gradient of the cost function can be computed using finite differences, this approach is usually slow (since it requires at least N function evaluations for each gradient evaluation). I use analytically determined gradients in all the algorithms presented later in this chapter.[2]

One strategy that can also be used to solve non-linear problems is the method of ***homotopy*** or the ***continuation method***. The continuation method involves solving a

[2]Verified by comparing with numerical gradients.

series of related problems of increasing difficulty that have similar solutions. Typically, the original problem is restated to include a parameter, such as β, which controls the difficulty of the problem. When $\beta = 1$ (for instance), the problem is easy to solve or has a analytical solution. When $\beta = 0$ (for instance) the original problem is recovered. To solve the original problem, a series of problems are solved while β is slowly decreased. The solution of each subsequent problem is used as the initial condition for the next problem.

One simple way to use the continuation technique is to introduce a regularization term based on β that creates a convex problem when $\beta = 1$,

$$min_u \sum_x \sum_y f(u) + \beta g(u). \tag{3.4}$$

where $g(u)$ is a regularization function that forces the problem to be convex.[3] During optimization, β is slowly reduced to zero, where the original problem is recovered. The hope with the continuation method is that the solutions to the series of problems will be close to the solution of the original problem so that they are good initial conditions.

I use a form of the continuation based method for the algorithms in this thesis. Since shape-from-shading problems have a well defined solution when singular points are in the image, the smoothness terms are not needed to guarantee a solution. The smoothness terms do, however, help convergence. During optimization, I slowly reduce the smoothness parameter toward zero in order to avoid biasing the solution [Horn and Brooks, 1986].

The questions of existence and uniqueness come up when working with optimization algorithms such as those presented in this thesis. For the types of cost functions presented in this thesis, it is clear that a solution exists; the cost functions are bounded from below by zero. That is, the best possible value for the cost function is zero and can be achieved only when the estimated surface images and the actual images match exactly and when any regularization terms are set to zero.

The uniqueness of a solution depends a great deal on the surface to be estimated. In general, both global and local minima will exist (as evidenced by the Hard Crater problem discussed in Chapter 7). The optimization techniques discussed above only guarantee convergence to a local minima. The global minimum may only be achieved if the initial conditions for the optimization algorithm are close to the true solution.

3.4 Speed-up Techniques

A very well researched part of optimization theory is how to speed up the convergence. For vision problems there are two promising speed up technologies: the use

[3]Convex problems have the property that the solution space is essentially bowl-like, with no local minima and a single global minima which makes them very easy to solve.

(Equations 2.19).

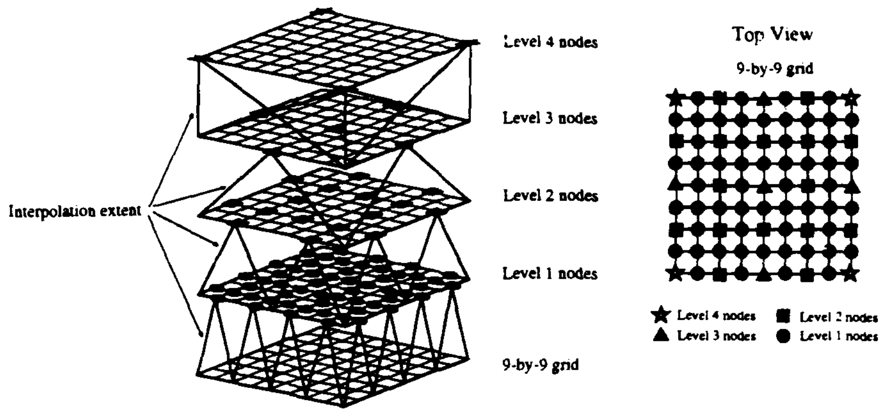


Figure 3-1: Hierarchical basis functions. Shown are the hierarchical basis nodes at each level (circles, triangles, etc.) and the associated interpolation function extent for the front-most nodes. The nodal basis for the 9-by-9 domain would have nodes at each pixel location and an extent half-again as small as the level 1 extents shown.

of *hierarchical basis functions*, and *multi-grid methods*. Both methods try to speed up the optimization problems by increasing the information transfer spatially. The methods are based on the property of many vision algorithms that an optimization variable within a grid of optimization variables may only be affected by its nearest neighbors. Due to the local connectedness, many vision algorithms have diffusion-like properties; the solution must diffuse throughout the grid. Schemes that directly transfer information over longer distances instead thus may speed up an algorithm.

Using hierarchical basis functions transforms the optimization space as seen by the optimization algorithm but not as seen by the vision algorithm. Basically it's like a change of basis. Figure 3-1 shows how a 9-by-9 domain would be represented in hierarchical basis. In particular, note that the nodes of the hierarchical basis propagate information over a much larger range (due to their extended interpolation extent) than the nodes in the nodal basis (particularly for the nodes at the upper levels). The figure shows linear interpolation between nodes, but any interpolation scheme can be used to build a hierarchical basis (see [Szeliski, 1990]). The problem is solved as before but with the new variables. If necessary, the variables can be transformed to the nodal basis when computing the cost function or gradient. Unfortunately, all these transformations have the potential to introduce round-off errors which can adversely affect sensitive algorithms.

The hierarchical basis functions have the most effect on the convergence and are the easiest to implement when the grid size is $2^n + 1$ where n is any positive integer. For such a grid it is possible to use $n + 1$ hierarchical basis levels. Using

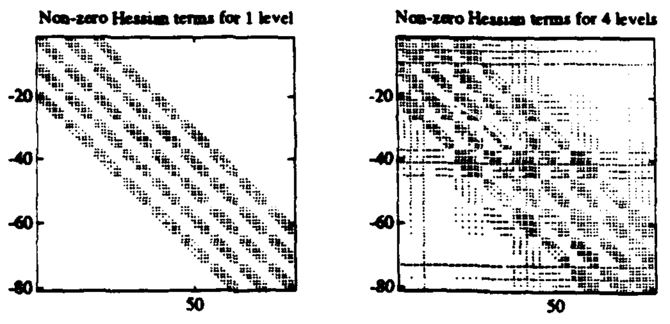


Figure 3-2: Non-zero Hessian elements for a 9-by-9 image. The plot on the left shows the non-zero Hessian elements when the nodal basis is used. The plot on the right shows the non-zero elements when a full hierarchical basis is used.

hierarchical basis functions increases the communication between nodes in the image array as shown in Figure 3-2. This increased communication speeds up the diffusion process considerably. I have noticed a five-fold increase in convergence for the photo-topography algorithms.

The multi-grid methods seek to propagate information over a larger range by solving a series of problems of different size. Usually the original problem is formulated on grids that decrease in size by a factor of two when going from one layer to the next (see Figure 3-3). The solutions on one layer are related to solutions on the layers above and below via interpolation or prolongation. The solutions are kept consistent with each other via both intra-layer and inter-layer processes (see [Terzopoulos, 1984] and [Brandt and Dinar, 1979]).

Multigrid methods have the potential to be much faster than the hierarchical basis functions since most of the computation (and optimization) is done on the smaller layers. Multigrid methods are well suited to linear problems (such as surface fitting), but may not work for non-linear problems. The problem is that typically

$$\sum g(u) \neq g(\sum u) \tag{3.5}$$

for non-linear functions $g(u)$, and the multi-grid methods require

$$\sum g(u) = g(\sum u) \tag{3.6}$$

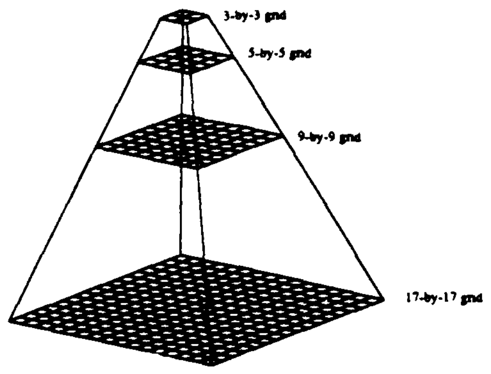


Figure 3-3: Multigrid levels

to constrain the solutions on the smaller grids so that they don't bias the solution on the larger grid.

One type of multi-grid method that can be used for non-linear problems is the *coarse-to-fine* method. In this method, the problem is solved on coarse layers first and the solution to each layer provides the initial condition to the next finer layer below. This method is significantly faster than just optimizing on the finest grid but doesn't produce as much convergence speed-up as the full multi-grid method.

Like the hierarchical basis methods, the multi-grid methods work best when the grid size is $2^n + 1$. However, since the multi-grid methods define a series of problems rather than just choosing a new set of basis functions, the multi-grid methods can be implemented easily for all grid sizes. There is some evidence, though, that grid size reductions should be near 2 for best convergence rates [Terzopoulos. 1984].

Chapter 4

Candidate algorithms

In this chapter I discuss several candidate algorithms for the photo-topography problem. Since each of the algorithms estimate the surface depth using a combination of shape-from-shading and stereo I call them *Depth From Shading and Stereo* (DFSS) algorithms. Along the way I will discuss the rationale behind each algorithm, its strengths and weaknesses, and how it fits in with the fusion techniques discussed so far. The performance of these algorithms on a set of synthetic test images is presented in Chapter 5. The performance of the z-only algorithm on real images is presented in Chapter 9.

The approach I am taking to fuse the vision algorithms is to develop a single cost function that incorporates the problem constraints along with some regularization terms to help direct the search path. This approach differs from many researchers' attempts at fusion algorithms in that it closely integrates the methods rather than building a module-based solution. I believe that this approach will create more robust solution methods.

In effect, I pose a generalized optimization problem which can be solved many ways. I have tried various solution methods with this research and have chosen direct optimization via the *conjugate gradient method* as my preferred method. The conjugate gradient method, like all direct optimization methods, guarantees reduction of the cost function at each step, in contrast to the more widely used Gauss-Seidel or Gauss-Jordan methods. The big advantage of the conjugate gradient method over other direct optimization methods is that no Hessian needs to be computed or stored. Computer vision algorithms have thousands of free variables which would result in a Hessian with millions of terms if it were computed.

I have looked at four basic algorithms for the photo-topography problem.

- The ***zpq*** and ***z*-only** algorithms estimate everything in a single global coordinate system that is defined to be halfway between the two camera positions. I refer to these algorithms as centralized. Figure 4-1 shows the flow of a typical centralized algorithm as a tree diagram. The current estimate of the surface height z is used to project points in the global coordinate system to points in

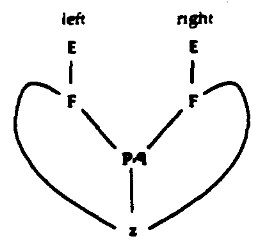


Figure 4-1: Centralized algorithm tree.

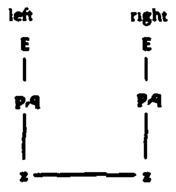


Figure 4-2: Decentralized algorithm tree.

each image (via perspective projection). These points won't in general land on a pixel center so some type of interpolation (e.g., bilinear or bicubic interpolation) is used to determine the value of the image at the projected points. This interpolated image F is then compared to a computed image based on the current estimate of the surface. The error is used to update p, q, and ultimately z. This type of algorithm closely integrates the constraints from both shading and stereo.

- The **dual-z** algorithm determines estimates of the surface for each image separately and takes into account the stereo information via a penalty term. I refer to this algorithm as decentralized. Figure 4-2 shows the flow of this type of algorithm. The estimated height for each image is used to compute the corresponding image estimate. The errors between this estimated image and the actual image for each camera are then used to update the associated p, q, and z for each image. So far this is the same as for a normal shape-from-shading algorithm. However, instead of just updating the surface height estimates directly from the image errors, the algorithm also takes into account the differences between the two surface estimates. This type of algorithm doesn't integrate the shading and stereo information as closely as the centralized algorithms.

- The **disparity** algorithm is also centralized algorithm but uses estimates of the disparity instead of the surface depth as the fundamental variable. Figure 4-3 shows the flow for this algorithm. The current estimate of the disparity u is used to project points in the global coordinate system to points in each image (via perspective projection). These points won't in general land on a pixel center

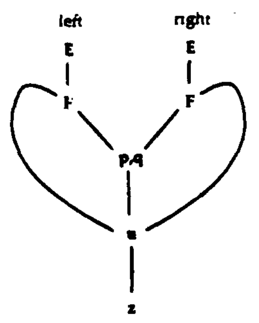


Figure 4-3: Centralized algorithm tree based on disparity.

so interpolation is used to determine the brightness value in the image at the projected points. This interpolated image F is then compared to a computed image based on the current estimate of the surface. The error is used to update p, q, and ultimately u. The surface height, z, is computed from u if needed. This type of algorithm also closely integrates the constraints from both shading and stereo.

The algorithms differ mostly in the representation of the surface topography. The representation chosen for the optimization variables is very important to the performance of computer vision algorithms. If the right representation is chosen then the search space will be smoother and contain less local minima than if the wrong representation is chosen.

4.1 *zpq* Algorithm.

This algorithm is based loosely on the Height and Gradient from Shading algorithm of Horn [Horn, 1989]. Following Horn's algorithm, surface height z, and the surface gradients p, and q, are used as optimization variables. The two photo-topography images, camera geometry, and surface reflectance functions are given as inputs to the algorithm. The equations that govern this situation are the photo-topography constraint equations (2.40) and the integrability constraint equations (2.39). The integrability constraint equations are necessary to ensure that p and q are consistent with the underlying surface z. The cost function for this algorithm is formed by integrating the squared error introduced by the current estimates for p, q, and z, together with a penalty functions for departure from integrability and departure from smoothness,

$$\min_{z,p,q} J = \frac{1}{2} \iint \left\{ \left(E^{(1)}(x + \frac{fb}{2z}, y) - R^{(1)}(p,q) \right)^2 + \left(E^{(2)}(x - \frac{fb}{2z}, y) - R^{(2)}(p,q) \right)^2 \right.$$

$$+ \mu \left[\left(\frac{f z_x}{x z_x + z} - p \right)^2 + \left(\frac{f z_y}{y z_y + z} - q \right)^2 \right]$$
$$+ \lambda \left[p_x^2 + p_y^2 + q_x^2 + q_y^2 \right] \Big\} \, dx \, dy. \tag{4.1}$$

This cost function allows the normal vector components p and q freedom to try to match the input images without requiring integrability. The penalty functions are then used to bias p and q toward solutions that are integrable and smooth. Using soft constraints like those above provides for more degrees of freedom which, it is hoped, will result in fewer local minima. The disadvantages of these extra degrees of freedom are a non-exact solution (since z will not be exactly consistent with p and q), and slower convergence.

The cost function above is continuous and must be discretized before it can be optimized (solved). The discretization process is an approximation process. The idea is to approximate the integral above by some function of a finite (usually small) set of variables. Researchers typically use either a series representation of the integrand or a sampled version of the integrand. Examples of the first representation are Fourier series and finite element methods. Examples of the second representation are finite difference methods.

I have used finite difference methods exclusively for the research presented in this thesis. The main reason for this is that the images ($E^{(i)}(x,y)$ above) are provided in digital form. In these images, each pixel represents, in some sense, a weighted average of the brightness falling within the sensitive area of the corresponding photosensor. Thus each image is actually an array of brightness values. Given this fact it makes sense to approximate the values for p, q, and z as arrays of gradient components or surface depth, as the case may be. Choosing this digital representation results in a problem of the finite difference class.

Suppose p, q, and z are represented as arrays, as discussed above. Suppose also that the continuous derivatives of the underlying functions (such as z) are approximated by finite differences, then the cost function can be approximated by the discrete sum,

$$\min_{p,q,z} J = \frac{1}{2MN\epsilon^2} \sum_{x,y \in \mathcal{D}} \Bigg\{ \left(F^{(1)}(x + \frac{fb}{2z}, y) - R^{(1)}(p,q) \right)^2$$
$$+ \left(F^{(2)}(x - \frac{fb}{2z}, y) - R^{(2)}(p,q) \right)^2$$
$$+ \mu \left[\left(p - \frac{f z_x}{x z_x + z} \right)^2 + \left(q - \frac{f z_y}{y z_y + z} \right)^2 \right]$$
$$+ \lambda \left[p_x^2 + p_y^2 + q_x^2 + q_y^2 \right] \Bigg\} \tag{4.2}$$

where $\mathcal{D}$ is the discrete domain of the underlying variables in the global coordinate

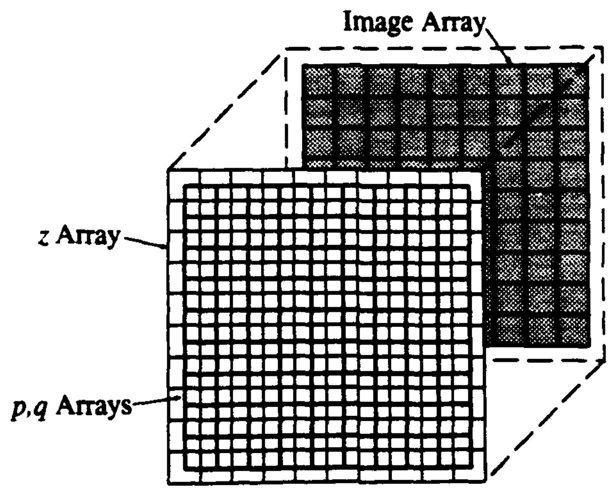


Figure 4-4: Relative sizes of the p, q, and z arrays in comparison to the image arrays for the co-grid representation.

system. M and N are the row and column dimensions of the discrete domain, and ϵ is the grid spacing (assumed to the same in both the x and y directions). The $F^{(i)}(x,y)$ are interpolated from the input images $E^{(i)}(x,y)$ using linear interpolation,

$$F^{(i)}(x \pm \frac{fb}{2z}, y) = E^{(i)}(\bar{x}, y) + (x \pm \frac{fb}{2z} - \bar{x})\left[E^{(i)}(\bar{x}+1, y) - E^{(i)}(\bar{x}, y)\right] \qquad (4.3)$$

where

$$\bar{x} = \text{floor}(x \pm \frac{fb}{2z}). \qquad (4.4)$$

The floor(x) function returns the greatest integer that is smaller than x. Note that Equation 4.3 assumes that x is sampled on a unity-spaced grid.

4.1.1 Co-grid Implementation

Two approaches have been taken to implement this cost function. In the first approach, the p and q arrays are the same size as the image arrays and the z array is one pixel larger in both the column and row directions. That is, the p and q functions are sampled on the *co-grid* of the function z (see Figure 4-4). This approach uses face-centered surface derivatives that are valid in the center of each 2-by-2 "face" formed by the z grid. In this approach, the x- and y-derivatives are approximated by computational molecules of the form,

$$(.)_x = \frac{1}{2\epsilon} \begin{array}{cc} \boxed{-1} & \boxed{1} \\ \boxed{-1} & \boxed{1} \end{array} \diamond (.), \tag{4.5}$$

and

$$(.)_y = \frac{1}{2\epsilon} \begin{array}{cc} \boxed{1} & \boxed{1} \\ \boxed{-1} & \boxed{-1} \end{array} \diamond (.) \tag{4.6}$$

$$\tag{4.7}$$

which have low approximation error. The diamond operator $\diamond$ is similar to (but different from) two dimensional convolution.[1] In addition, this approach requires that z be sampled on the co-grid when computing the terms in the cost function that contain z. Based on a bilinear approximation, z can be sampled using

$$\bar{z} = \frac{1}{4} \begin{array}{cc} \boxed{1} & \boxed{1} \\ \boxed{1} & \boxed{1} \end{array} \diamond z. \tag{4.9}$$

The resulting algorithm performs as well as the algorithm to be described next but has the disadvantage that the hierarchical basis functions cannot be used to full advantage since not all the p, q, and z arrays can be of the size $2^n + 1$. Recall that the hierarchical basis functions are easiest to implement and provide for optimum communication between pixels in the array when the row and column dimensions are of size $2^n + 1$ for some n. In addition, multi-grid methods also cannot be used when the variable arrays are of different sizes.

4.1.2 Matched-grid Implementation

In the second approach, p, q, and z are chosen to all be the same size as the image arrays. In such a representation, all the functions are sampled on the same grid (see Figure 4-5) which is why this approach is called the *matched-grid representation*. This approach uses vertex-centered surface derivatives that are valid at each vertex of the

[1]Suppose $h(i,j)$ is a computational molecule (as an array), then the diamond operation is defined as

$$(h \diamond z)(i,j) = \sum_{k,m} z(i - M + k, j - N + m)h(k,m) \tag{4.8}$$

where M is the row dimension of h and N is the row dimension of h. The definition differs from two-dimensional convolution in that h is not "flipped".

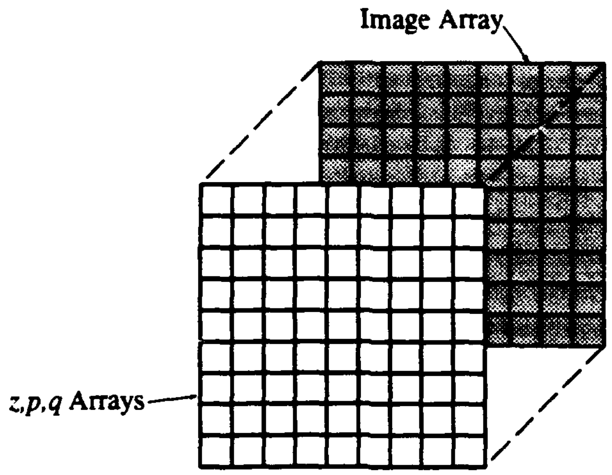


Figure 4-5: Relative sizes of the p, q, and z arrays in comparison to the image arrays for the matched-grid representation.

z grid. Computational molecules that exemplify this approximation are,

$$(.)_x = \frac{1}{2\epsilon} \boxed{-1} \; \boxed{0} \; \boxed{1} \diamond (.), \tag{4.10}$$

$$(.)_y = \frac{1}{2\epsilon} \begin{array}{c} \boxed{1} \\ \boxed{0} \\ \boxed{-1} \end{array} \diamond (.) \tag{4.11}$$

which have higher approximation error than their face-centered counterparts. In practice, the algorithm based on this approach has nearly the same performance as the algorithm based on the face-centered approach. Since p and q are the same size as z, some type of approximation must be made at the array edges (boundaries). I have chosen to extrapolate the estimates using a bicubic interpolant and use this extrapolated version of each estimate in subsequent calculations (see Section A.2). Using the extrapolated estimate is similar to using natural boundary conditions in variational calculus methods, and allows for easy computation of the gradient.

The performance of the matched-grid zpq algorithm on four test image sets is presented in Chapter 5. This algorithm gets stuck in a local minimum on the hard crater images (as do all of the algorithms to be presented), but does reasonable well on the other test images (except for the mountain images). As a whole, this algorithm produces surfaces that are too smooth and takes longer to converge than the other algorithms.

Like all the algorithms to be presented, this algorithm is run using a exponentially changing smoothness parameter λ, and integrability parameter μ (if it exists). A large initial smoothness parameter helps to make the search space more bowl-like at the expense of meeting the photo-topography constraints. When the smoothness parameter is small, the photo-topography constraints are met at the expense of a possibly bumpy surface. A small initial integrability parameter allows the surface gradient components p and q to wander far from integrability in order to create image estimates that match the input images. A large integrability parameter. requires that these image estimates be created so that p and q are nearly integrable.

While it is not required that the optimization parameters be ramped in this fashion, I have found that doing so allows for faster convergence over optimizing with the parameters fixed at their final values.

4.2 z-only Algorithm

This next cost function is similar to the zpq cost function except that it uses hard integrability constraints instead of soft constraints. With hard constraints we are guaranteed that any solutions obtained will be feasible. The trade-off is that the algorithm will have less degrees of freedom to work with and may be more susceptible to local minima.

For this cost function, the depth map z is the only optimization variable thus the resulting algorithm is called the *z-only algorithm*. The surface gradient components p and q are computed directly from the depth map. As before, the photo-topography images, camera geometry, and surface reflectance functions are inputs to the algorithm. The constraint equations again come from the photo-topography constraints and integrability constraints. The cost function is formed by integrating the squared photo-topography error introduced by the current estimate for z, together with a penalty function for departure from smoothness,

The penalty function is mainly used to guide the solution towards the minimum. In practice, the smoothness weighting parameter, λ, is slowly reduced toward zero as the algorithm converges.

$$\min_z J = \frac{1}{2}\iint\left\{\left(E^{(1)}(x+\frac{fb}{2z},y) - R^{(1)}(p,q)\right)^2 + \left(E^{(2)}(x-\frac{fb}{2z},y) - R^{(2)}(p,q)\right)^2 \right.$$
$$\left. + \lambda\left[z_{xx}^2 + 2z_{xy}^2 + z_{yy}^2\right]\right\} dx\,dy. \tag{4.12}$$

The smoothness term is based on what is called the second variation. It is equivalent to $p_x^2 + p_y^2 + q_x^2 + q_y^2$ when $z_x \approx fp/z_0$ and $z_y \approx fq/z_0$ where z_0 is the nominal depth.[2]

[2]The approximations are valid when the field of view is small, the image is centered around the camera's principal point, and the depth of field relative to the nominal depth is small.

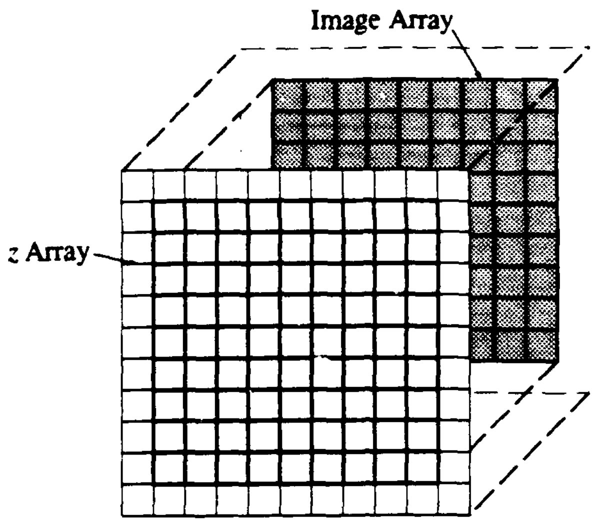


Figure 4-6: Relative size of the z array in comparison to the image arrays for the expanded boundary implementation.

Using an array for z, the discrete approximation of this cost function is,

$$\min_{z} J = \frac{1}{2MN\epsilon^2} \sum_{x,y\in\mathcal{D}} \left\{ \left(F^{(1)}(x + \frac{fb}{2z}, y) - R^{(1)}(p,q) \right)^2 + \left(F^{(2)}(x - \frac{fb}{2z}, y) - R^{(2)}(p,q) \right)^2 + \lambda \left[z_{xx}^2 + 2z_{xy}^2 + z_{yy}^2 \right] \right\}. \tag{4.13}$$

The $F^{(i)}(x \pm \frac{fb}{2z}, y)$ are interpolated from the input images $E^{(i)}(x,y)$ as in the previous algorithm.

4.2.1 Expanded Boundary Implementation

Two approaches have been taken to implement this cost function. In the first approach, the z array is 1 pixel larger than the image array in all directions (see Figure 4-6). The normal vector components are estimated on the inner grid which is the same size as the image arrays. In fact, the outer depth estimates are only used when computing p and q. This approach is very easy to implement but cannot be used with a multi-grid scheme since the z array is a different size than the image arrays. The approach can, however, take full advantage of hierarchical basis functions since z can be of the size $2^n + 1$. The implementation uses the vertex-centered derivatives discussed in the previous algorithm. The second-order derivatives are approximated by the following computational molecules which are also vertex-centered approximations.

$$z_{xx} = \frac{1}{4\epsilon^2} \begin{array}{ccc} \boxed{1} & \boxed{-2} & \boxed{1} \\ \boxed{2} & \boxed{-4} & \boxed{2} \\ \boxed{1} & \boxed{-2} & \boxed{1} \end{array} \diamond z, \tag{4.14}$$

$$z_{xy} = \frac{1}{\epsilon^2} \begin{array}{cc} \boxed{-1} & \boxed{1} \\ \boxed{1} & \boxed{-1} \end{array} \diamond z, \tag{4.15}$$

$$z_{yy} = \frac{1}{4\epsilon^2} \begin{array}{ccc} \boxed{1} & \boxed{2} & \boxed{1} \\ \boxed{-2} & \boxed{-4} & \boxed{-2} \\ \boxed{1} & \boxed{2} & \boxed{1} \end{array} \diamond z. \tag{4.16}$$

(4.17)

The algorithm based on this approach has nearly the same performance as the algorithm based on the approach to be presented next.

4.2.2 Matched Grid Implementation

In the second approach, z is chosen to be the same size as the image arrays thus facilitating a multi-grid implementation. The surface normal components, p and q, are computed on the boundary using the boundary extrapolation technique discussed for the zpq algorithm. This approach also uses vertex-centered approximations for the derivatives.

The performance of the matched-grid z-only algorithm on the four test images is presented in Chapter 5. This algorithm also gets stuck in the local minimum for the hard crater images. The performance on the easy crater images and the others is very good however. In fact, this algorithm has the best performance of the all the algorithms to be presented. An important thing to note in the performance figures is that the algorithm converges to a reasonable surface quite quickly, usually in the first 100 iterations or so. The performance figures are shown for 1200 iterations so that it is possible to see the convergence characteristics in the long term.

4.3 Dual-z Algorithm

This cost function uses a totally different variable representation. In this case the depth map is represented by two arrays, $z^{(1)}$ and $z^{(2)}$ which are estimated based on the images $E^{(1)}$ and $E^{(2)}$ respectively. This algorithm is referred to as the *dual-z* algorithm. The two depth arrays are kept consistent with one another using a penalty function that enforces the stereo constraint equations,

$$\begin{aligned} z^{(1)}(x^{(1)}, y^{(1)}) &= z^{(2)}(x^{(1)} - \frac{fb}{z^{(1)}}, y^{(1)}), \\ z^{(2)}(x^{(2)}, y^{(2)}) &= z^{(1)}(x^{(2)} + \frac{fb}{z^{(2)}}, y^{(2)}) \end{aligned} \tag{4.18}$$

where $(x^{(1)}, y^{(1)})$ are the coordinates of points in the first image and $(x^{(2)}, y^{(2)})$ are the coordinates of points in the second image. These equations require that the depth estimate from one image be the same as the depth estimate at the corresponding projected point[3] in the other image. Each depth map is a constrained shape-from-shading solution for the corresponding image. This algorithm has a totally different search space than any of the previous algorithms.

For this cost function, the depth maps $z^{(i)}$ are the optimization variables and the surface gradients $p^{(i)}$ and $q^{(i)}$ are computed directly from the $z^{(i)}$. The photo-topography images, camera geometry, and the surface reflectance functions are inputs. The constraints come from the photo-topography constraints, the integrability constraints, and the stereo constraints. The cost function is formed by integrating the squared shading error (in the spirit of shape-from-shading) introduced by the current estimates of $z^{(i)}$, together with penalty functions for departure from smoothness, and stereo error.

$$\begin{aligned} \min_{z^{(1)}, z^{(2)}} J = \frac{1}{2} \iint \Bigg\{ & \left(E^{(1)}(x^{(1)}, y^{(1)}) - R^{(1)}(p^{(1)}, q^{(1)})\right)^2 + \lambda(\nabla^2 z^{(1)})^2 \\ & + \mu \left(z^{(1)}(x^{(1)}, y^{(1)}) - z^{(2)}(x^{(1)} - \frac{fb}{z^{(1)}}, y^{(1)}) \right)^2 \Bigg\} dx^{(1)}\, dy^{(1)} \\ + \frac{1}{2} \iint \Bigg\{ & \left(E^{(2)}(x^{(2)}, y^{(2)}) - R^{(2)}(p^{(2)}, q^{(2)})\right)^2 + \lambda(\nabla^2 z^{(2)})^2 \\ & + \mu \left(z^{(1)}(x^{(2)} + \frac{fb}{z^{(2)}}, y^{(2)}) - z^{(2)}(x^{(1)}, y^{(2)}) \right)^2 \Bigg\} dx^{(2)}\, dy^{(2)} \end{aligned} \tag{4.19}$$

where

$$\nabla^2 z = z_{xx} + z_{yy}. \tag{4.20}$$

The smoothness penalty function for this cost function is based on the squared Lapla-

[3]Using perspective projection.

cian of $z^{(i)}$. The squared Laplacian has roughly the same characteristics as the second variation [Grimson, 1979], but is much quicker to compute (since it only involves one convolution).

The cost function contains two shape-from-shading cost functions and the stereo penalty function. This cost function is not as closely-coupled as the other two cost functions I have described so far. In particular the stereo constraints are added on like an addendum to the shape-from-shading cost function. The stereo constraints, in effect, pass their constraints up to the shape from shading parts of the algorithm. I think of this algorithm as trying to fit together two rubber mountains. The mountains can be moved back and forth, and molded to find the best fit. Since the algorithm is not as integrated, we would expect that images that don't contain strong shading information will be difficult for this algorithm to solve.

Using $z^{(1)}$ and $z^{(2)}$ arrays as before, the cost function can be discretized to become.

$$\begin{aligned}\min_{z^{(1)},z^{(2)}} J = \frac{1}{2MN\epsilon^2} \sum_{x^{(1)},y^{(1)}\in\mathcal{D}^{(1)}} \Bigg\{ & \left(E^{(1)}(x^{(1)},y^{(1)}) - R^{(1)}(p^{(1)},q^{(1)})\right)^2 + \lambda(\nabla^2 z^{(1)})^2 \\ & + \mu\left(z^{(1)}(x^{(1)},y^{(1)}) - \bar{z}^{(2)}(x^{(1)} - \frac{fb}{z^{(1)}}, y^{(1)})\right)^2 \Bigg\} \\ + \frac{1}{2MN\epsilon^2} \sum_{x^{(2)},y^{(2)}\in\mathcal{D}^{(2)}} \Bigg\{ & \left(E^{(2)}(x^{(2)},y^{(2)}) - R^{(2)}(p^{(2)},q^{(2)})\right)^2 + \lambda(\nabla^2 z^{(2)})^2 \\ & + \mu\left(\bar{z}^{(1)}(x^{(2)} + \frac{fb}{z^{(2)}}, y^{(2)}) - z^{(2)}(x^{(1)},y^{(2)})\right)^2 \Bigg\}\end{aligned} \tag{4.21}$$

The stereo penalty functions include terms involving $\bar{z}^{(1)}$ and $\bar{z}^{(2)}$ which are interpolated versions of $z^{(1)}$ and $z^{(2)}$. Bicubic interpolation is used [Keys, 1981]. Using this notation, $\bar{z}^{(i)}(x,y)$ is the value of $z^{(i)}$ evaluated at the point (x,y) using bicubic interpolation. Vertex-centered derivative approximations were used with this cost function, and the Laplacian is approximated using the computational molecule,

$$\nabla^2 z = \frac{1}{6\epsilon^2} \begin{array}{ccc} \boxed{1} & \boxed{4} & \boxed{1} \\ \boxed{4} & \boxed{-20} & \boxed{4} \\ \boxed{1} & \boxed{4} & \boxed{1} \end{array} \diamond z. \tag{4.22}$$

Both the expanded boundary approach and the matched-grid approach discussed in the previous section were taken to implement this cost function. Several variations of the basic implementation were tried with this cost function. For example, in addition to the smoothness term based on the squared Laplacian, algorithms were implemented where the smoothness term was based on the second variation.

$$\min_{z^{(1)},z^{(2)}} J = \frac{1}{2MN\epsilon^2} \sum_{x^{(1)},y^{(1)}\in\mathcal{D}^{(1)}} \left\{\cdots + \lambda\left[(z_{xx}^{(1)})^2 + 2(z_{xy}^{(1)})^2 + (z_{yy}^{(1)})^2\right]\cdots+\right\}$$
$$+ \frac{1}{2MN\epsilon^2} \sum_{x^{(2)},y^{(2)}\in\mathcal{D}^{(2)}} \left\{\cdots + \lambda\left[(z_{xx}^{(2)})^2 + 2(z_{xy}^{(2)})^2 + (z_{yy}^{(2)})^2\right]\cdots+\right\} \quad (4.23)$$

Another alternate implementation used face-centered derivatives instead of vertex-centered derivatives. All of these algorithms have performance very similar to each other, and they suggest that as long as a reasonable derivative approximation or smoothness criteria is used, it doesn't really matter which one is chosen. Any reasonable approximation or smoothness criteria is as good as any other. In light of this, it makes sense to choose the approximations or smoothness criteria that are the easiest and the fasted to compute, or that have the best numerical properties.

The performance of this algorithm on four test images is presented in Chapter 5. Like the previous algorithm, this algorithm also has problems with the hard crater images but does well on the other images. Unfortunately, the results of this algorithm are two depth maps which represent the depths as seen from the right and left cameras. These maps will not, in general, register exactly with one another. To create a single depth map from these maps requires some type of averaging. This is a major disadvantage.

To address this disadvantage, an algorithm was implemented that had three depth maps $z^{(1)}$, $z^{(2)}$, and z, where z is the depth map as seen from the global coordinate system. In all other ways the algorithm was the same as the dual-z algorithm. The performance of the algorithm was very similar to the performance of the above algorithm yet it created a central depth map. It is not shown here since it was even slower than the dual-z algorithm.

The combination of two depth maps and need for bicubic interpolation, caused the dual-z algorithm to be the slowest to compute. The figures in Chapter 5 show the number of function evaluations and not the computation time. In practice this algorithm was 3-5 times slower than the z-only algorithm.

4.4 Disparity Map Algorithm

Another popular representation for depth that shows up often in the vision literature is the disparity map. The disparity is the relative offset of a point in one image with respect to the corresponding point in the other image. Using the notation we have used so far, the disparity can be defined as.

$$u = \frac{fb}{2z}, \quad (4.24)$$

where u is the disparity. Using this definition of disparity, the surface gradient components are computed using,

$$\begin{aligned} p &= \frac{fu_x}{xu_x - u}, \\ q &= \frac{fu_y}{yu_y - u}. \end{aligned} \tag{4.25}$$

The photo-topography constraint equations can also be re-written using disparity to become,

$$\begin{aligned} E^{(1)}(x+u,y) &= R^{(1)}(p,q), \\ E^{(2)}(x-u,y) &= R^{(2)}(p,q). \end{aligned} \tag{4.26}$$

Suppose we adopt the centralized approach for this cost function and have the disparity map u be the optimization variables. The surface gradients p and q are computed directly from u. The photo-topography images, camera geometry, and surface reflectance functions are inputs. The cost function for this case is formed by integrating the squared photo-topography error (based on disparity) together with a penalty function for departure from smoothness,

$$\begin{aligned} \min_u J = \frac{1}{2} \iint \Big\{ & \left(E^{(1)}(x+u,y) - R^{(1)}(p,q)\right)^2 + \left(E^{(2)}(x-u,y) - R^{(2)}(p,q)\right)^2 \\ & + \lambda \left[u_{xx}^2 + 2u_{xy}^2 + u_{yy}^2\right]\Big\}\, dx\, dy. \end{aligned} \tag{4.27}$$

where the surface gradient components are given by Equation 4.25. The smoothness term is based on the second variation of u. The main difference between this cost function and the others I have presented is the smoothness term (it is now based on disparity instead of depth). The numerical properties will also be somewhat different since the hierarchical basis functions will be over disparity instead of depth.

Disparity is typically used in vision algorithms for two reasons. First the disparity-based algorithms usually have fewer division operations which can improve performance, and second the disparity has a much smaller dynamic range which can contribute to better numerical properties. Unfortunately, since the photo-topography problem requires the use of perspective projection, the first reason doesn't hold for my algorithms. The culprits are the surface gradient equations (Equations 4.25). As for the second reason, the disparity-based DFSS algorithm has pretty much the same numerical properties as the other DFSS algorithms. However, using disparity, instead of depth, changes the solution space the optimization is performed in.

Representing u as an array, the cost function can be discretized to obtain,

$$\begin{aligned} \min_u J = \frac{1}{2MN\epsilon^2} \sum_{x,y\in\mathcal{D}} \Big\{ & \left(F^{(1)}(x+u,y) - R^{(1)}(p,q)\right)^2 + \left(F^{(2)}(x-u,y) - R^{(2)}(p,q)\right)^2 \\ & + \lambda \left[u_{xx}^2 + 2u_{xy}^2 + u_{yy}^2\right]\Big\} \end{aligned} \tag{4.28}$$

The $F^{(i)}(x \pm u, y)$ are linearly interpolated from the input images $E^{(i)}(x, y)$ using disparity,

$$F^{(i)}(x \pm u, y) = E^{(i)}(\bar{x}, y) + (x \pm u - \bar{x})\left[E^{(i)}(\bar{x}+1, y) - E^{(i)}(\bar{x}, y)\right] \qquad (4.29)$$

where

$$\bar{x} = \text{floor}(x \pm u). \qquad (4.30)$$

Only the matched-grid approach was used to implement this cost function. In this case, u is chosen to be the same size as the image arrays. The derivatives are approximated using vertex-centered computational molecules as for the previous algorithms.

The performance of this algorithm on four test images is presented in Chapter 5. This algorithm, like all the previous algorithms, gets stuck in a local minimum for the hard crater images. The algorithm, however, performs well on the other images. An interesting observation can be made about the performance of this algorithm on the mountain images. The algorithm over-estimates the mountain height while the z-only algorithm under-estimates it. This is clear indication that the search space for the disparity-based algorithm is fundamentally different from the search space for the z-only algorithm. Unfortunately, the solution space is not different enough to avoid falling into the local minimum of the hard crater image.

Chapter 5

Test Results

It is important when developing a new algorithm to be able to test its performance. For computer vision algorithms, that means that it must be possible to compare the estimated surface with the actual surface. The only way to do this with complete confidence in the results is to create synthetic images from a known surface topography, run the algorithm on these images, and compare the estimate with the known topography. Only after it is shown that the algorithm works on the synthetic images, can we be confident that the surface estimates for actual planet images will be accurate.

I have created four image pairs as test images based on three surface topologies.

- **Easy Crater Images.** This set of images is based on a crater on a flat plane. The light sources are oblique which results in images that prove to be easy for the DFSS algorithms to interpret.

- **Hill Images.** This set of images is based on a fractally-generated set of rolling hills. The light sources are oblique. These images are interpreted correctly for most of the DFSS algorithms but require more iterations of each algorithm since they are more complicated than the crater images.

- **Mountain Images.** This set of images is based on a fractally-generated mountainous terrain. The light sources are oblique and the camera baseline is much smaller. This set of images poses a challenge to the DFSS algorithms due to the steep terrain and reduced baseline.

- **Hard Crater Images.** This set of images is also based on a crater on a flat plane. The light sources are almost directly behind the camera resulting in a set images that prove to be difficult for the DFSS algorithms.

The calibration parameters for the test images are summarized in Table 5-1. The table lists values for the baseline distance b, camera focal distance f, nominal depth z_0, and light source vector components $p_S^{(1)}$, $q_S^{(1)}$, $p_S^{(2)}$, $q_S^{(2)}$. The focal distance and

	b	f	z_0	$\Delta z/z_0$	Δu	$p_S^{(1)}$	$q_S^{(1)}$	$p_S^{(2)}$	$q_S^{(2)}$
Easy crater	500	-2750	-997	0.0038	2.62	0.2	-0.5	-0.3	0.1
Hills	500	-12222	-1000	0.0011	3.41	1.0	-1.0	0.3	0.1
Mountain	100	-2292	-996	0.0153	1.77	0.5	0.5	-0.5	0.0
Hard crater	500	-2750	-997	0.0038	2.62	0.1	0.1	-0.1	0.1

Table 5-1: Camera geometry.

nominal depth are negative to be consistent with a right handed coordinate system. The baseline distance b and depth z have the same units (say miles), while the camera focal distance, pixel spacing and the light source components are based on camera units (say millimeters). The table also lists values for relative surface height $\Delta z/z_0$ and disparity Δu. These two parameters indicate the difficulty of the problem. Small values of either indicate a hard problem. Note the units for Δu are in pixels for a 65-by-65 problem, and f is computed so that the pixel spacing is 1 (millimeter).

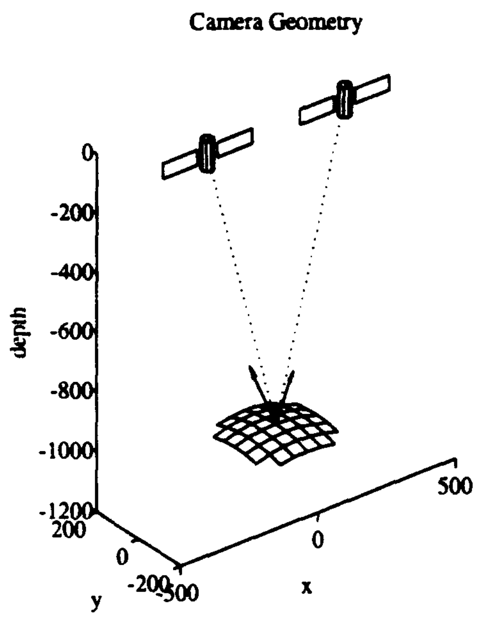
All of the test images are noise-free so that the best performance of each algorithm can be tested. The performance of the z-only algorithm on images with noise in presented in Chapter 7. The performance figures for each test case include mesh plots of the estimated surface at several points during optimization as well as estimated images based on the final surface. The mesh plots are of a 33-by-33 smoothed (to avoid aliasing) and subsampled version of the 65-by-65 estimated surface. The lower resolution mesh plots are used to avoid printing problems.

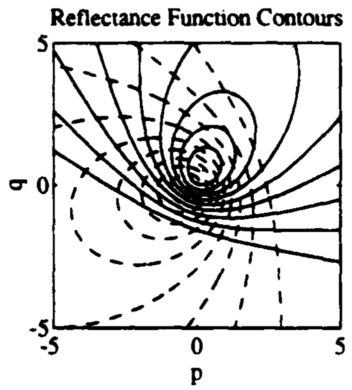
The performance figures also show the number of function evaluations (iterations) computed by each algorithm during convergence. The number of conjugate gradient updates taken is between one-half to one-third of the number of function evaluations since 2-3 function evaluations are needed to perform each update step. All of the algorithms use hierarchical basis functions to enhance convergence.

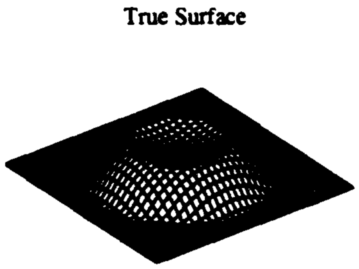
5.1 Easy Crater Images

The crater on a flat plane is very simple surface and thus serves as a good test surface.[1] The camera and light source geometry as well as the true surface. light source contour plots and resulting images are shown in Figure 5-1. As shown in the figure. the baseline distance between the cameras is about half the distance to the surface and the light source (i.e., the sun) positions for the two images differ greatly from each other. As a result, this set of images has strong shading information which is very easy for the DFSS algorithms to take advantage of. The effects of the lighting can be seen in the contour plots of the reflectance map. The reflectance contours are separated enough in gradient space (p-q space) so that brightness values from

[1]Information on how to generate this surface can be found in Appendix C.







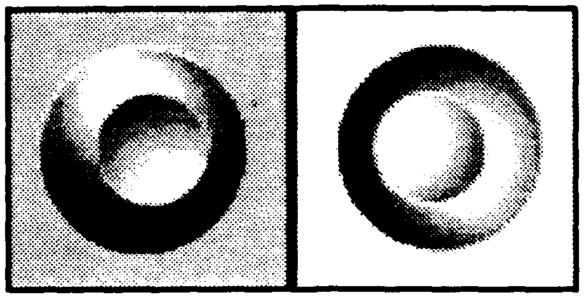


Figure 5-1: Test images of crater on flat plane (easy case). Shown are the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, reflectance function contours for the two light source positions, and the left and right synthetic noise-free images.

one image can easily constraint the possible set of feasible gradient directions.[2] The images are 65-by-65 pixels in size and are noise free.

The Figures 5-2–5-5 show the results of applying the various DFSS algorithms to this test case. All four algorithms can correctly interpret these images but the z-only algorithm performs best. The figures show that the estimated images in this case very closely resemble the true images and the surface estimates are very good except in the lower left corner of the surface (see the mesh plots). With the lighting conditions chosen for these test images, this small anomaly has little effect on the estimated images. It is interesting to note that the anomaly shows up for all four algorithms.

The most interesting thing about this test case is the rate of convergence that is obtained with the z-only, dual-z, and disparity algorithms. In those cases, a pretty good estimate is obtained after only 50 function evaluations! This represents only about 20 updates since more than one function evaluation is necessary per update when using conjugate gradient optimization.

[2]That is, there are regions in the gradient space where a given brightness value from one image strongly constrains the gradient direction there.

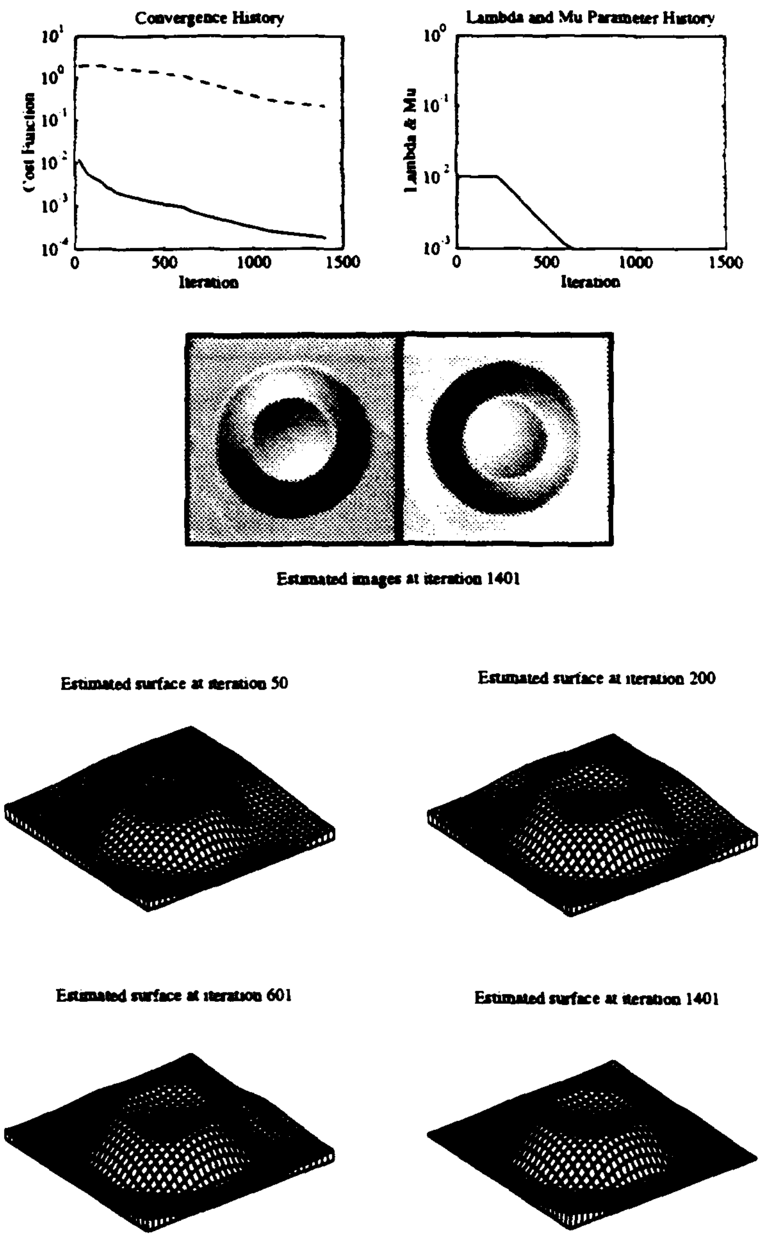


Figure 5-2: Performance of the *zpq* algorithm on the easy crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

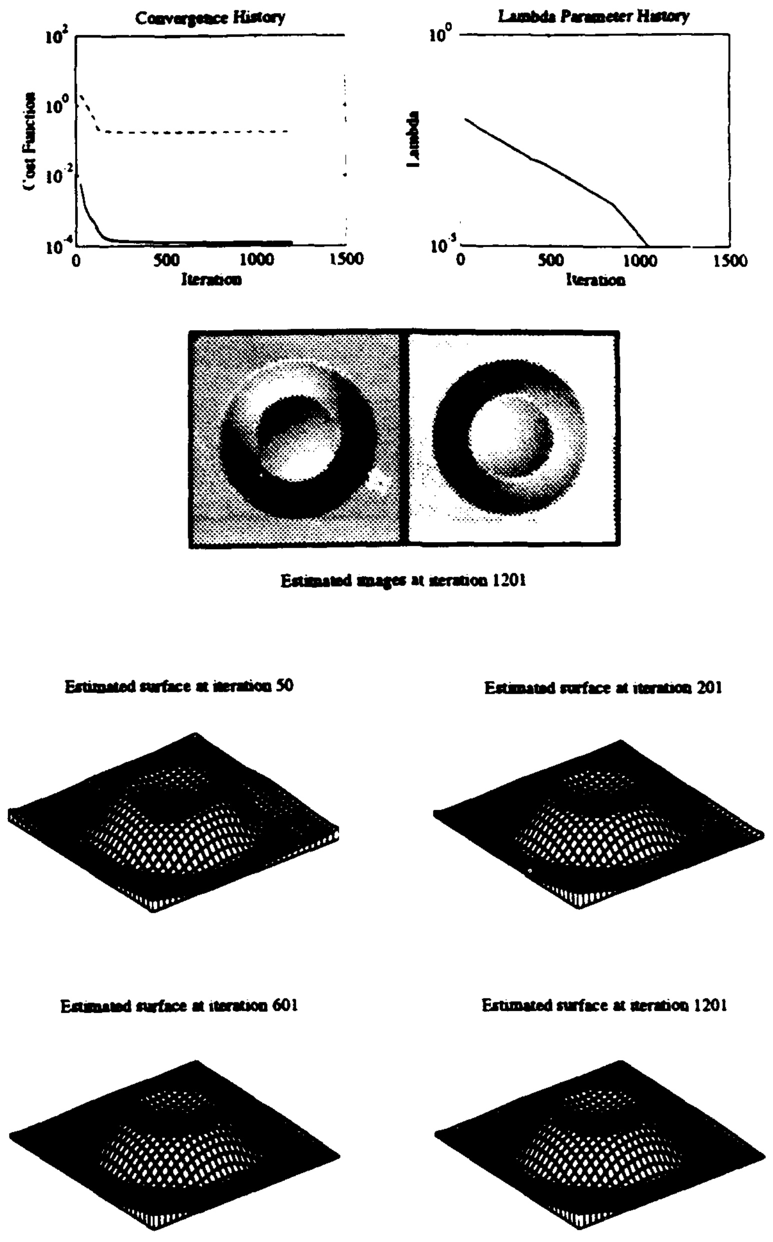
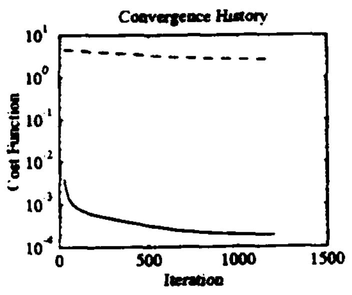
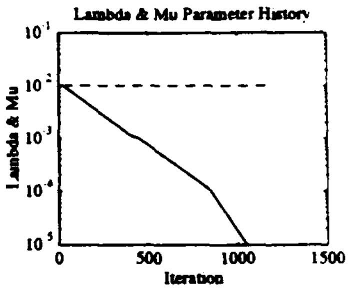
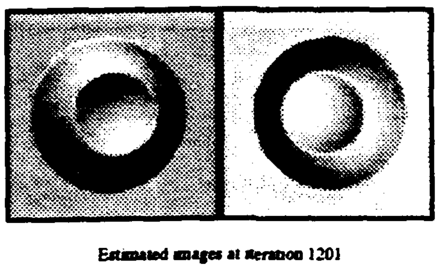
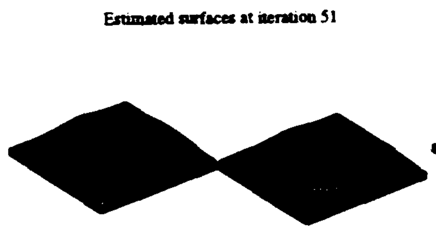


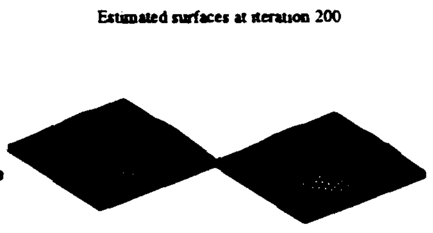
Figure 5-3: Performance of the z-only algorithm on the easy crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

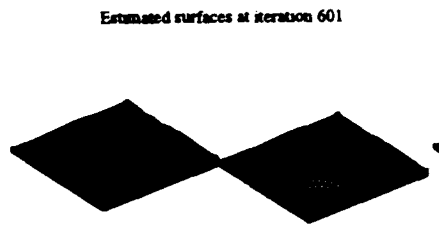












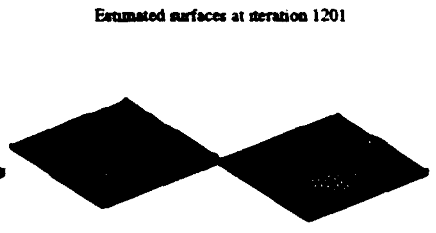


Figure 5-4: Performance of the dual-z algorithm on the easy crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surfaces and mesh plots of the surfaces at different points during optimization.

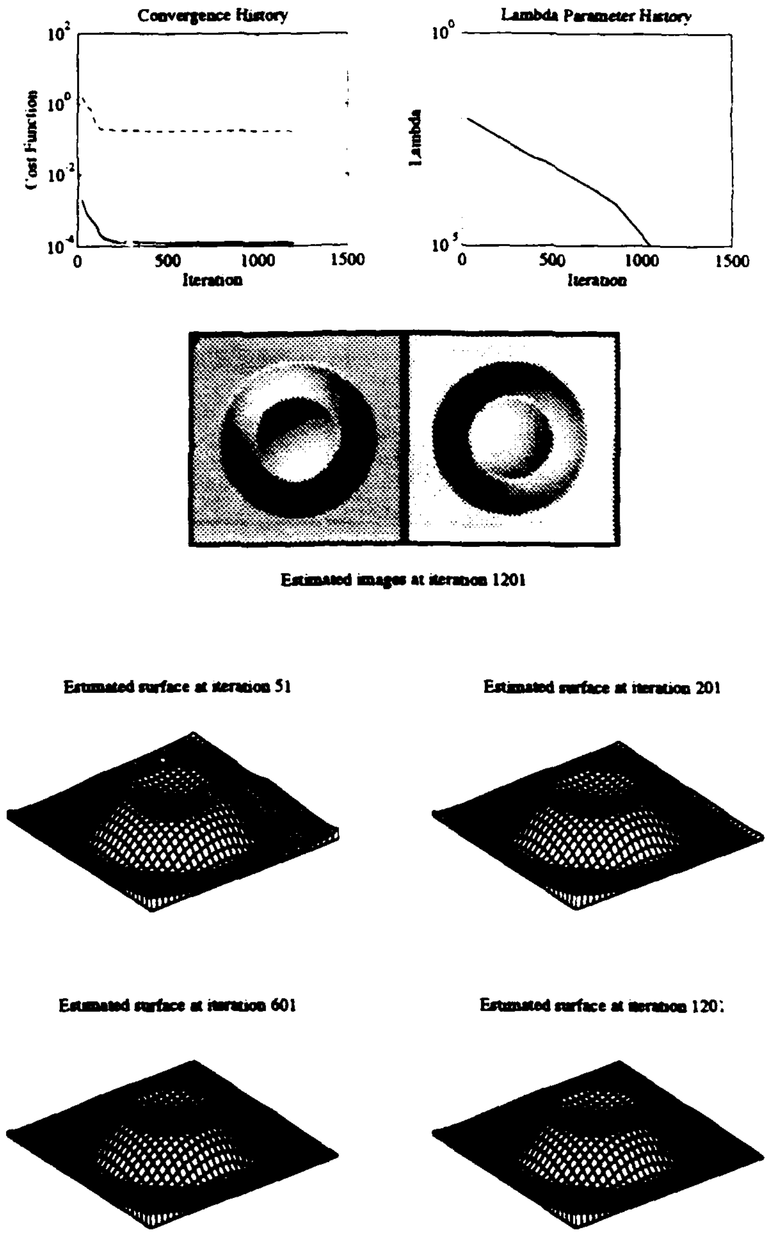


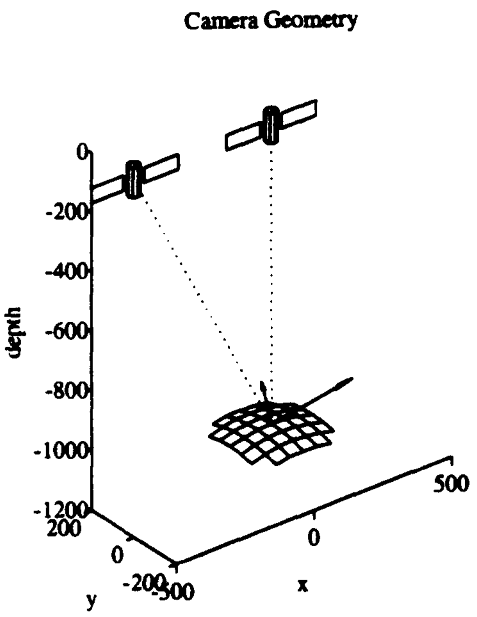
Figure 5-5: Performance of the disparity-based algorithm on the easy crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

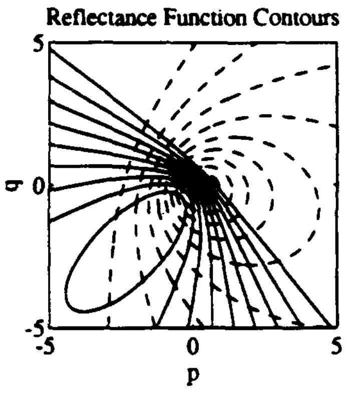
5.2 Hill Images

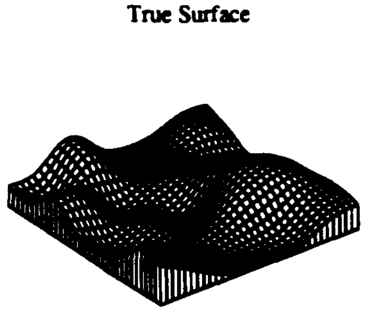
The second set of images is of an undulating surface similar to eroded hills and was generated using a fractal technique.[3] Figure 5-6 shows the camera geometry, true surface, reflectance contours, and test images for this surface. This set of images is more representative of the type of terrain the DFSS algorithms are likely to encounter. As shown in the figure, the left camera is directly over the surface and the right camera views the surface obliquely. The light sources are separated, as for the easy crater image, resulting in images with strong shading information. I find the relatively low 65-by-65 resolution of these images hard to interpret visually (unlike the crater images which are easy to interpret). The DFSS algorithms, however, perform well with these images.

The Figures 5-7–5-10 show the results of applying the various DFSS algorithms to this test case. All four algorithms work reasonably well with these images, but the z-only algorithm performs best. Due to the complexity of the surface, this set of images requires more iterations of each algorithm to obtain a satisfactory estimate of the surface than the easy crater images. The figures show that the z-only algorithm performs best, it even obtains a very good estimate after 50 iterations! The convergence is a little slower for the dual-z and disparity algorithms; they obtain good estimates after about 200 iterations. The zpq algorithm is the slowest and requires about 600 iterations to obtain a satisfactory solution. The estimated images in all cases match very well the true images shown in Figure 5-6.

[3]Information on how to generate this surface can be found in Appendix C.







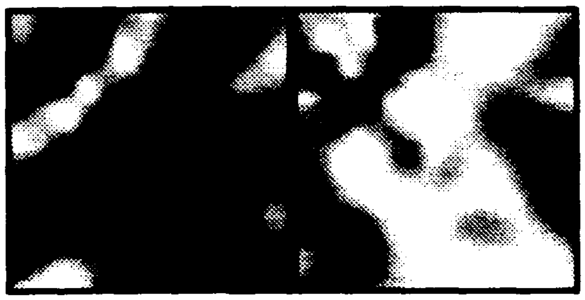
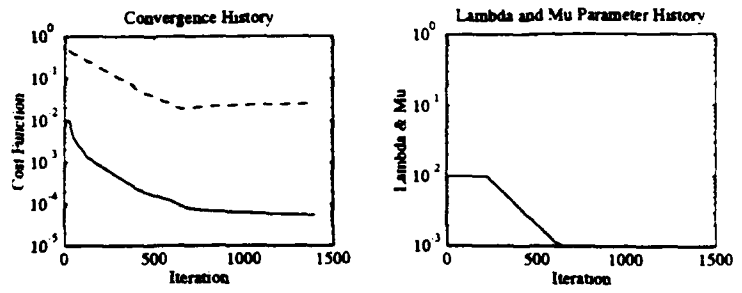
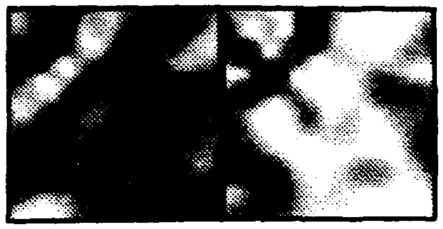


Figure 5-6: Test images of hill. Shown are the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, reflectance function contours for the two light source positions, and the left and right synthetic noise-free images.





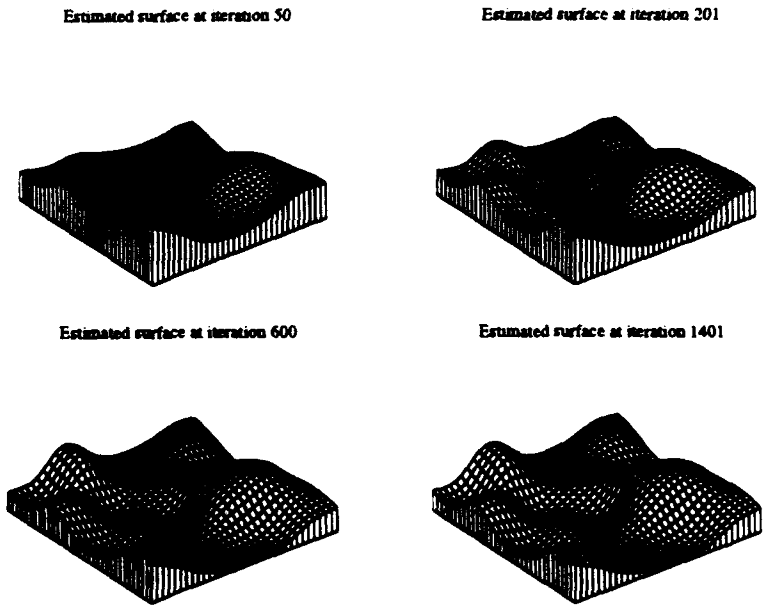

Estimated surface at iteration 50
Estimated surface at iteration 201
Estimated surface at iteration 600
Estimated surface at iteration 1401

Figure 5-7: Performance of the *zpq* algorithm on the hill images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

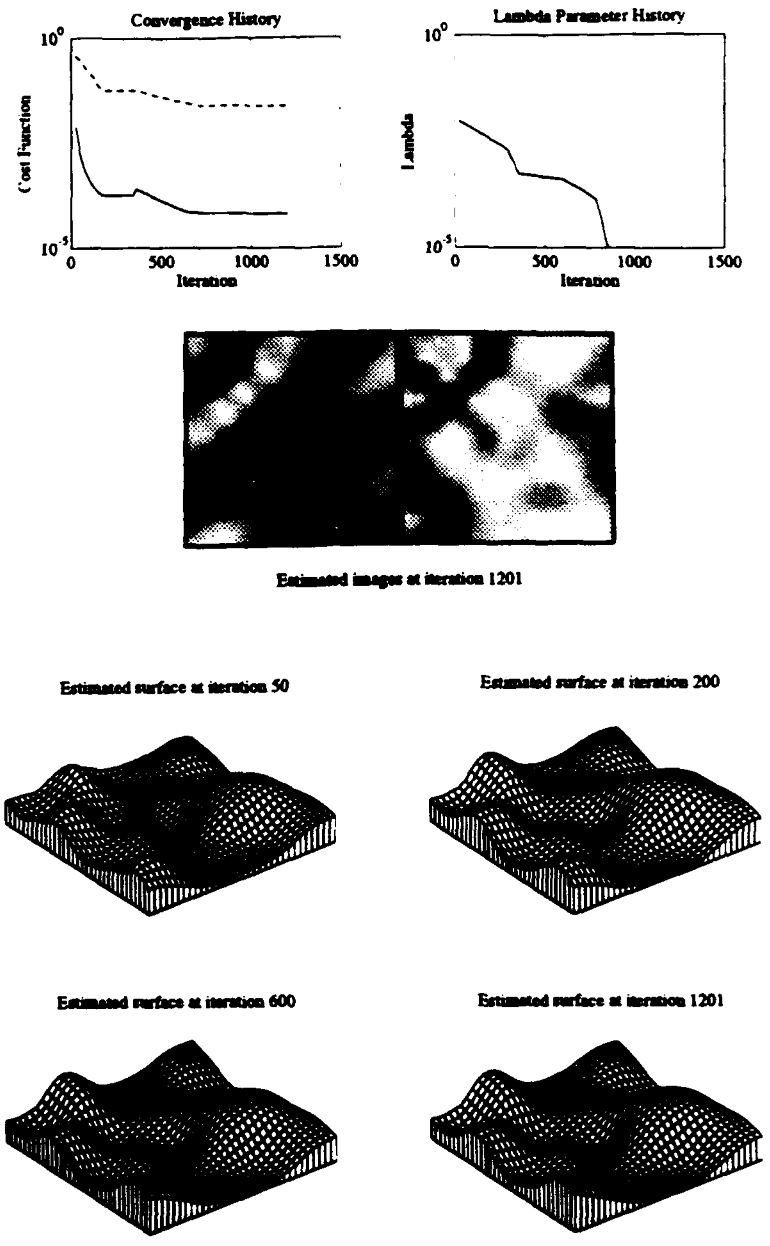
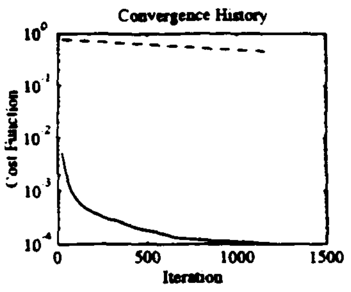
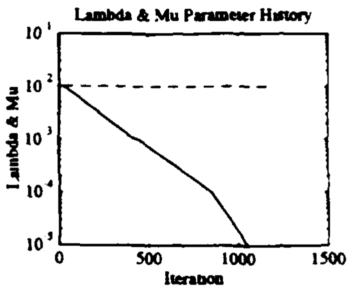
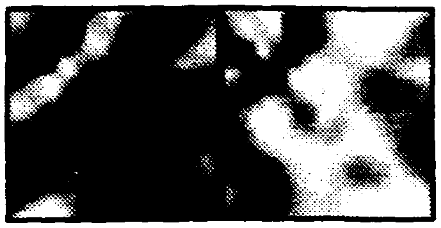


Figure 5-8: Performance of the z-only algorithm on the hill images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.







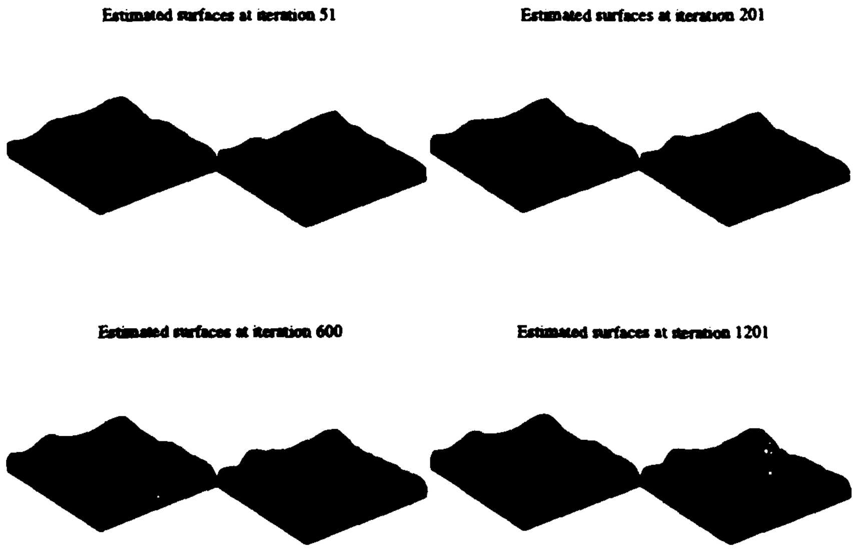

Estimated surfaces at iteration 51
Estimated surfaces at iteration 201
Estimated surfaces at iteration 600
Estimated surfaces at iteration 1201

Figure 5-9: Performance of the dual-z algorithm on the hill images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surfaces and mesh plots of the surfaces at different points during optimization.

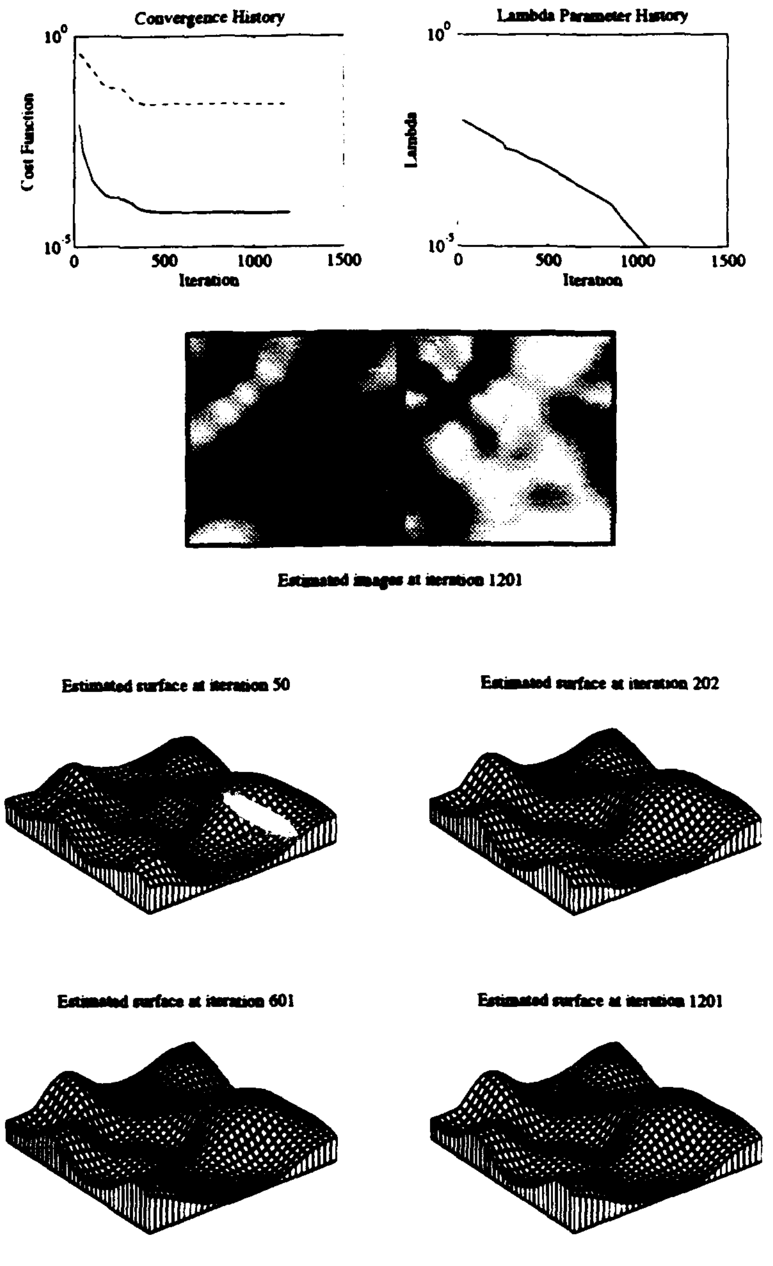
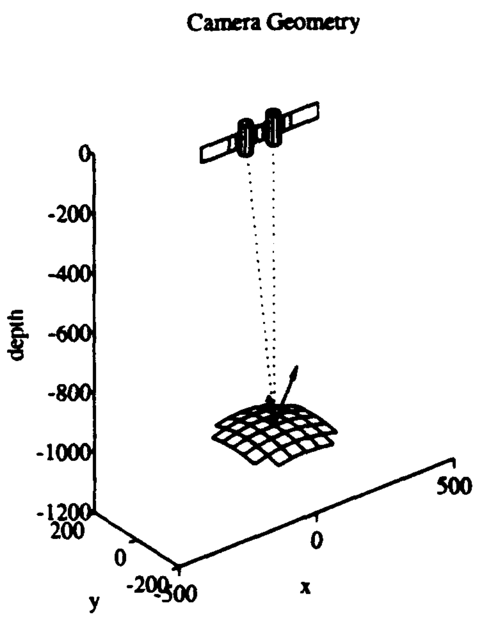


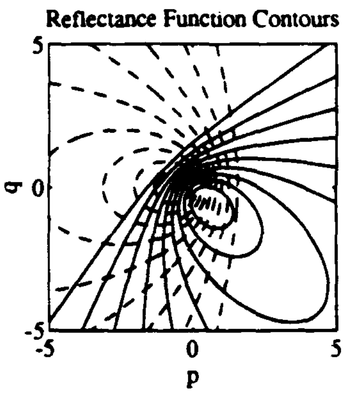
Figure 5-10: Performance of the disparity-based algorithm on the hill images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

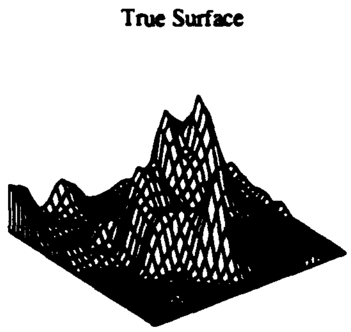
5.3 Mountain Images

The third set of images is of a highly mountainous surface and was created to show the performance of the algorithms on steep terrain. The steep terrain requires that the baseline distance from the cameras be shortened so that all the surface points are visible from both cameras (see Figure 5-11). Thus this set of images can be used to test the performance of the algorithms when the stereo baseline is small. As shown in the figure, the light source positions for this set of images are widely separated and generate deep shadows on this steep terrain. The reflectance maps are flat within a shadow so no helpful gradient is available to the algorithms. In addition, knowledge that a particular pixel is in shadow only constrains the set of possible gradient directions to a sub-plane of gradient space. Thus, within a shadow region, much more influence is given to the brightness values from the other image. This combination of effects results in slower convergence.

The Figures 5-12–5-15 show the results of applying the various DFSS algorithms to this test case. The z-only, dual-z, and disparity algorithms work reasonably well with these images, but the z-only algorithm performs best. The zqq algorithm has trouble with this test case. Due to the complexity of this test case, the algorithms require many more iterations to achieve a satisfactory solution. The z-only and disparity algorithms are able to obtain good solutions after about 600 iterations, while 1400 iterations are not enough for the zpq. The dual-z algorithm obtains a reasonable estimate at about 1200 iterations.







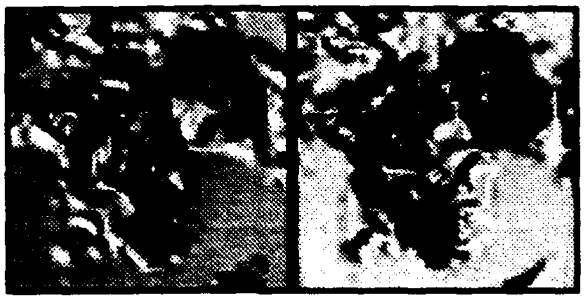


Figure 5-11: Test images of mountain. Shown are the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, reflectance function contours for the two light source positions, and the left and right synthetic noise-free images.

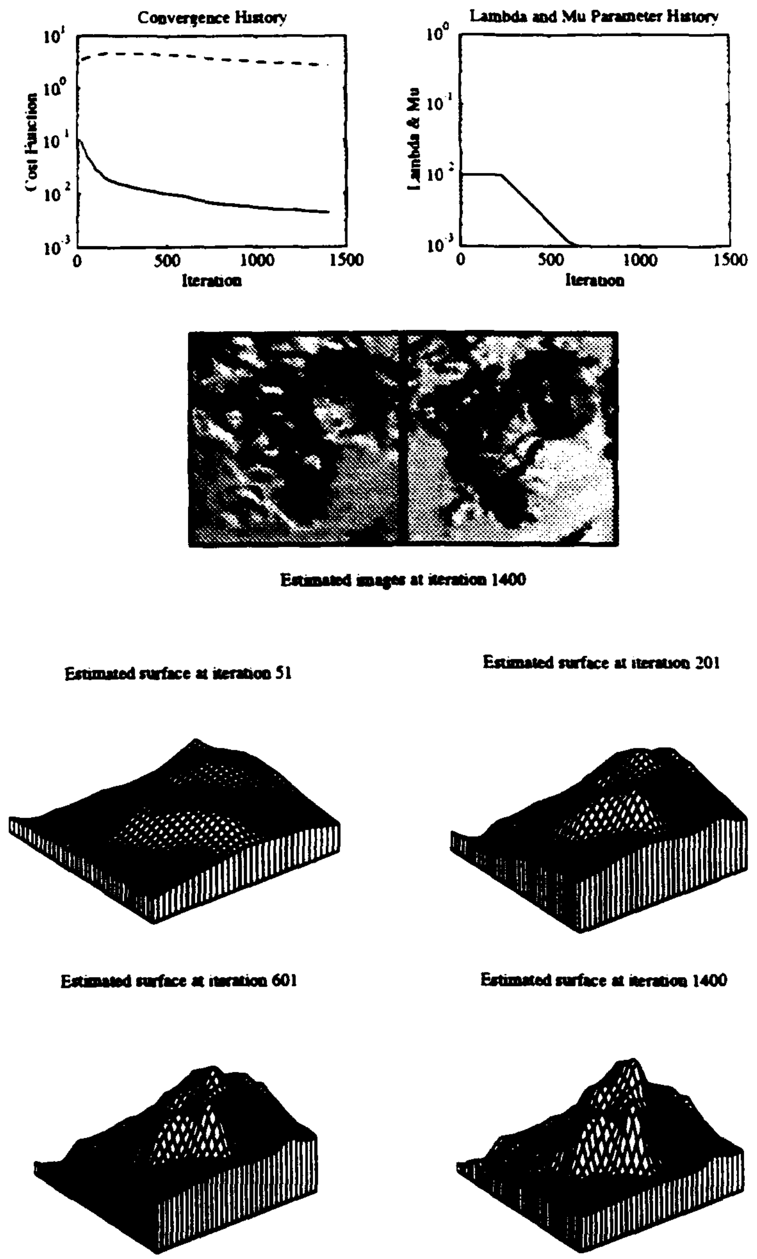
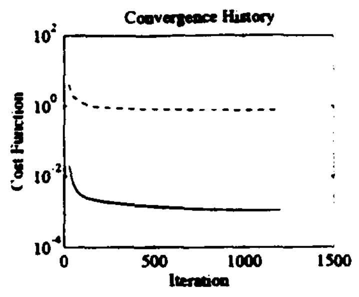
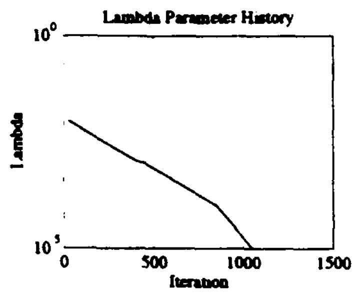
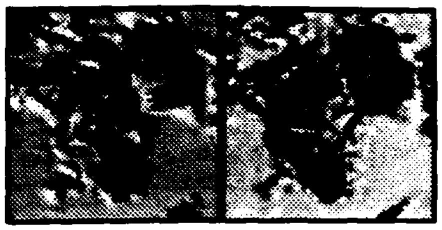


Figure 5-12: Performance of the *zpq* algorithm on the mountain images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

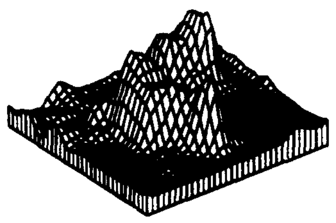




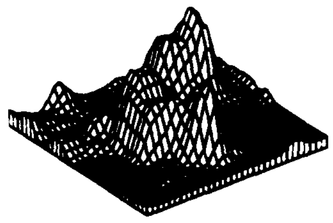


Estimated images at iteration 1200

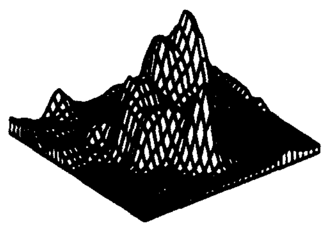
Estimated surface at iteration 51



Estimated surface at iteration 200



Estimated surface at iteration 600



Estimated surface at iteration 1200

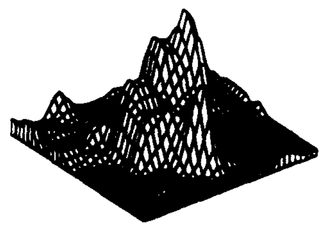
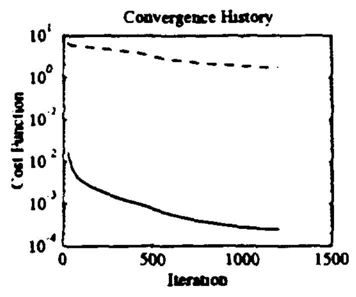
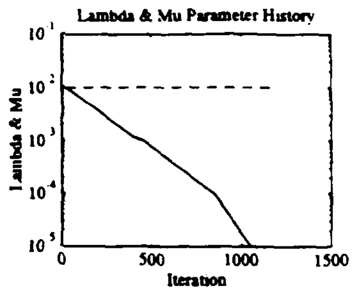
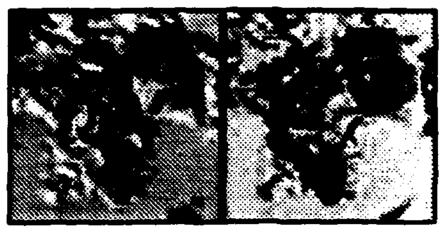
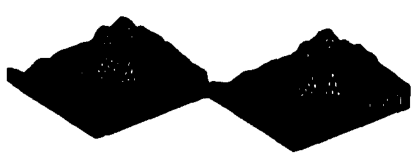


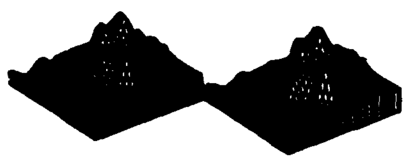
Figure 5-13: Performance of the z-only algorithm on the mountain images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

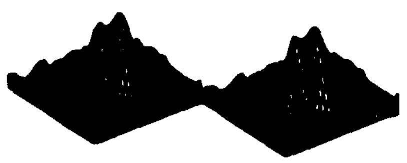












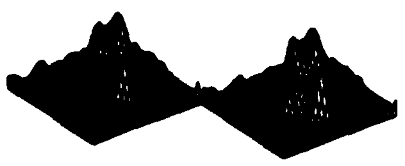
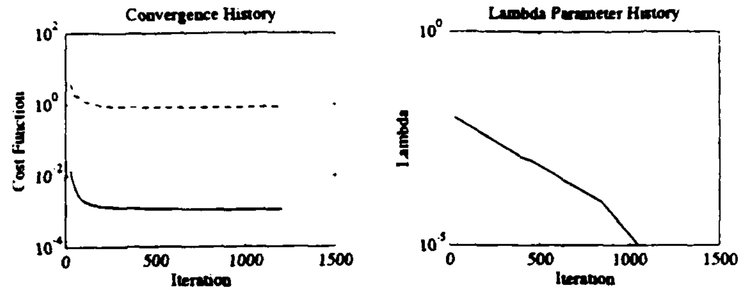
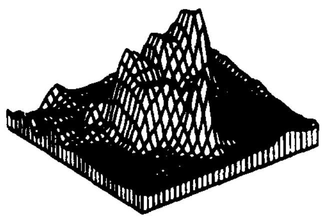


Figure 5-14: Performance of the dual-z algorithm on the mountain images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surfaces and mesh plots of the surfaces at different points during optimization.

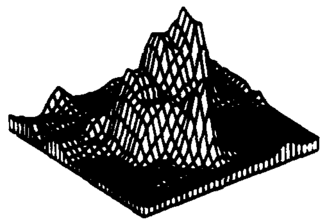


Estimated images at iteration 1201

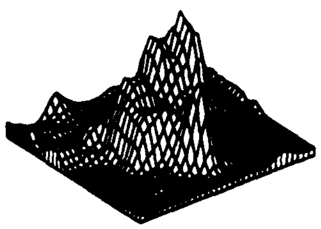
Estimated surface at iteration 51



Estimated surface at iteration 201



Estimated surface at iteration 601



Estimated surface at iteration 1201

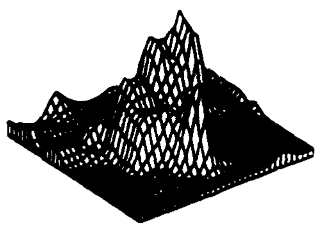


Figure 5-15: Performance of the disparity-based algorithm on the mountain images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

5.4 Hard Crater Images

Like the easy crater images presented earlier, this set of images is also based on the crater on a flat plane. However in this case, the light source positions are nearly the same and are almost directly behind the cameras. The resulting images are very bright and look very much the same except for slight differences in shading (see Figure 5-16). The reflectance maps for this case are nearly co-incident which contributes to a surface orientation ambiguity. This set of images represents a worst case for the DFSS algorithms since the shading information in the images is weak and the range of brightness values is small. However, they also have strong stereo correspondence information, which unfortunately is not readily utilized by the DFSS algorithms. The test images are 65-by-65 pixels in size and are noise free.

The Figures 5-17–5-20 show the results of applying the various DFSS algorithms to this test case. All four algorithms get stuck in a local minimum. The figures show that the estimated images resemble the true images of Figure 5-16 even though the estimated surface doesn't match the actual surface. The problem is that the algorithms incorrectly interpret the surface as concave when it is actually convex (see also the shape-from-shading results in Chapter 6). Even though the stereo information in these test images can be used to correctly determine the orientation of the surface, the DFSS algorithms rely very heavily on the shading information.

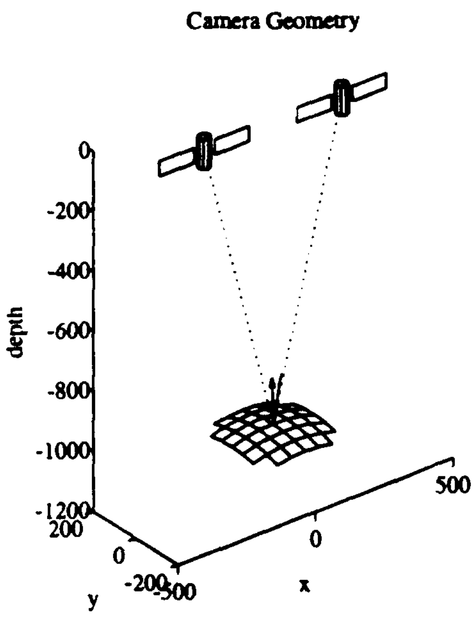
The surface orientation ambiguity is a result of having both light sources directly behind the cameras. All the test images that I have tried that have this light source geometry cause the algorithms to fail.

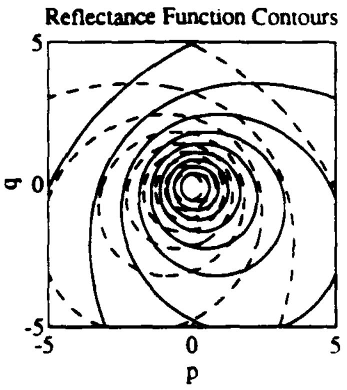
The results shown in the figures are based on algorithms that employ the hierarchical basis functions. When I run the algorithms on the hard crater test images without using the hierarchical basis functions, the algorithms can correctly interpret the images. The convergence is significantly slower, however. Based on several runs I have made, it appears that the explanation for this behavior is that the hierarchical basis functions lead the algorithm down a particular path that ends up in the local minima. When the nodal basis is used, the algorithm goes down a different path that is able to bypass the local minimum (just barely).

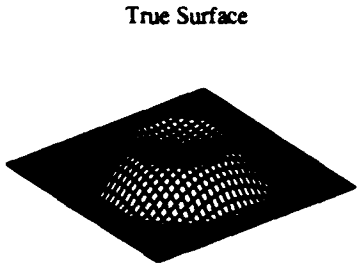
This problem seems to be sensitive to initial condition. In fact, when I start the algorithms from an initial condition close to the true surface, they do converge to the true solution as expected.[4]

This test case has been the impetus for developing a plethora of algorithms, only a few of which are presented in this thesis. I had hoped that developing an algorithm with a totally different search space would result in one that could solve this problem. Unfortunately, all the algorithms I have developed have gotten stuck in the same local minima shown in the figures. The other test cases show that the algorithms perform very well when different lighting is used where at least one lighting condition is oblique.

[4]Of course, they only achieve the exact solution when the smoothness parameters are set to zero.







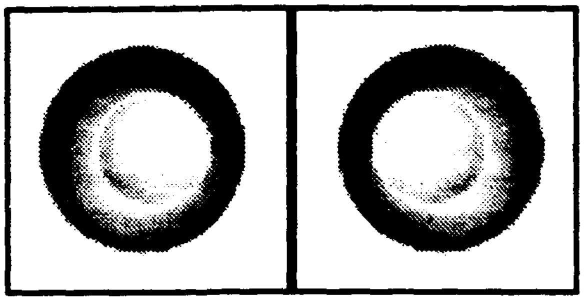


Figure 5-16: Test images of crater on flat plane (hard case). Shown are the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, reflectance function contours for the two light source positions, and the left and right synthetic noise-free images.

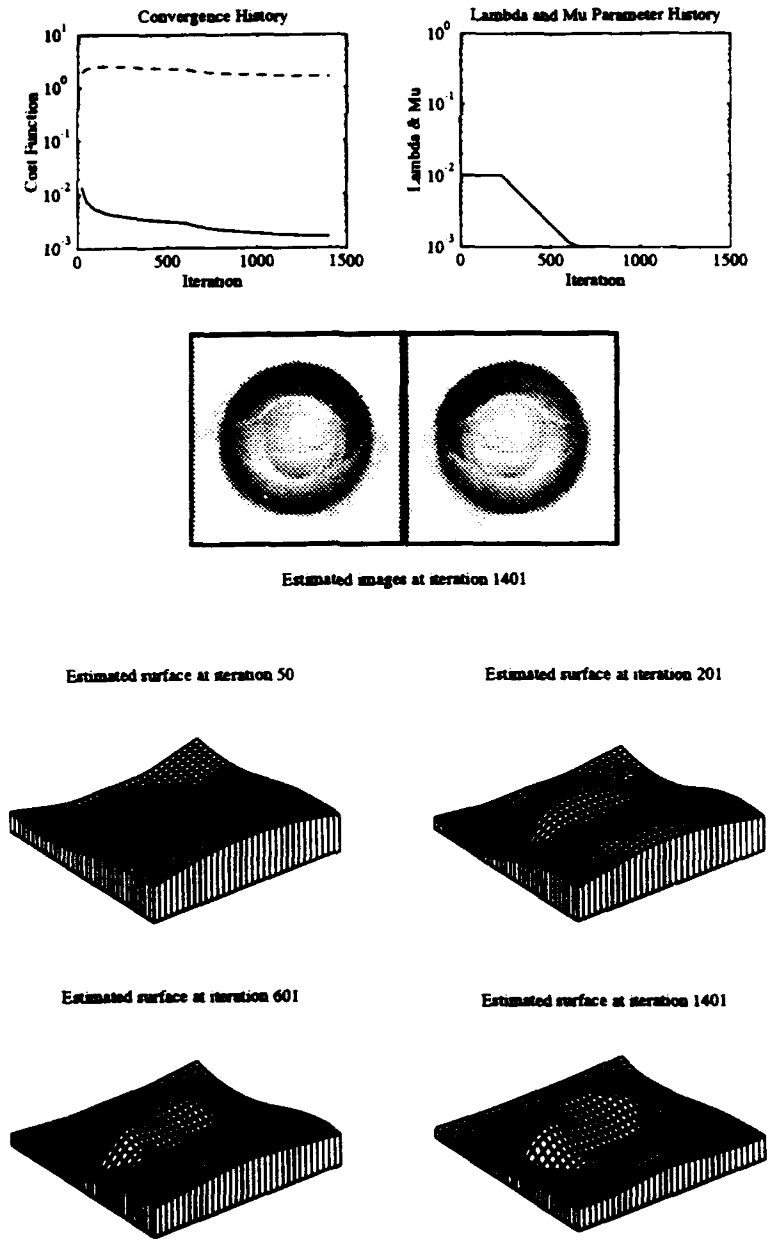


Figure 5-17: Performance of the *zpq* algorithm on the hard crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

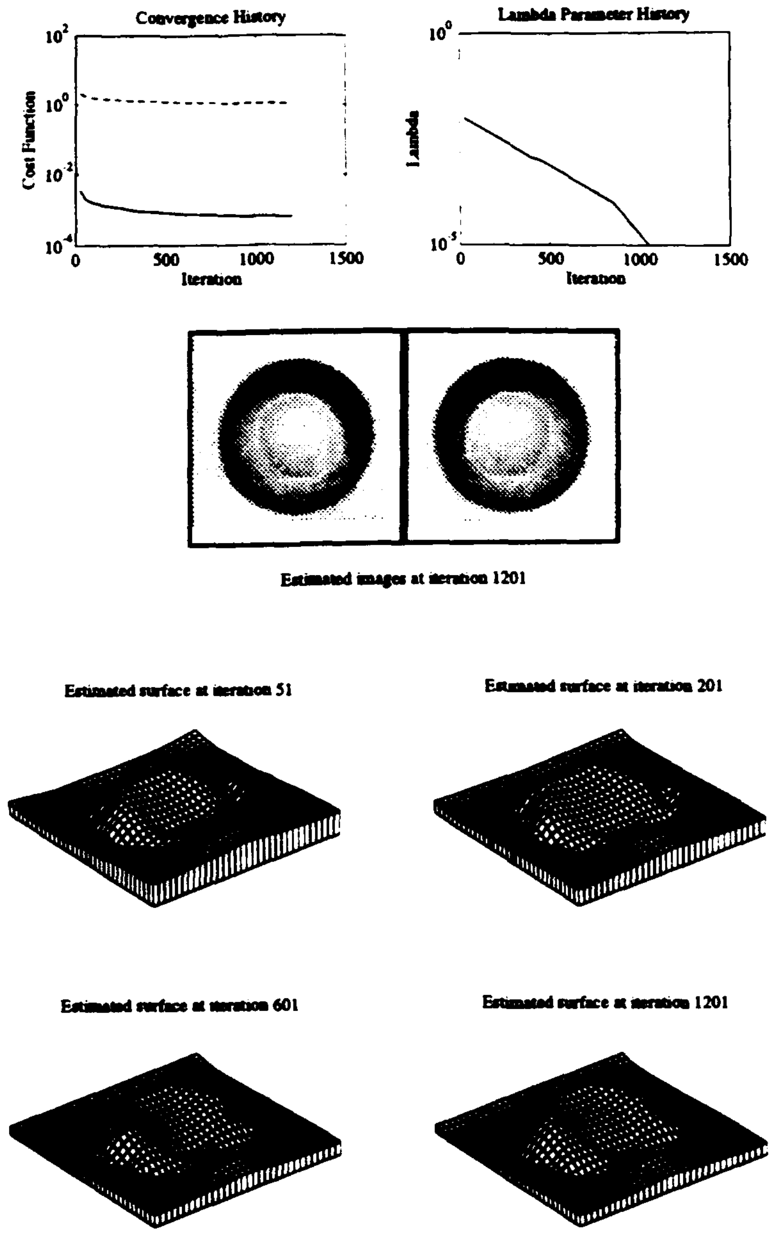


Figure 5-18: Performance of the z-only algorithm on the hard crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

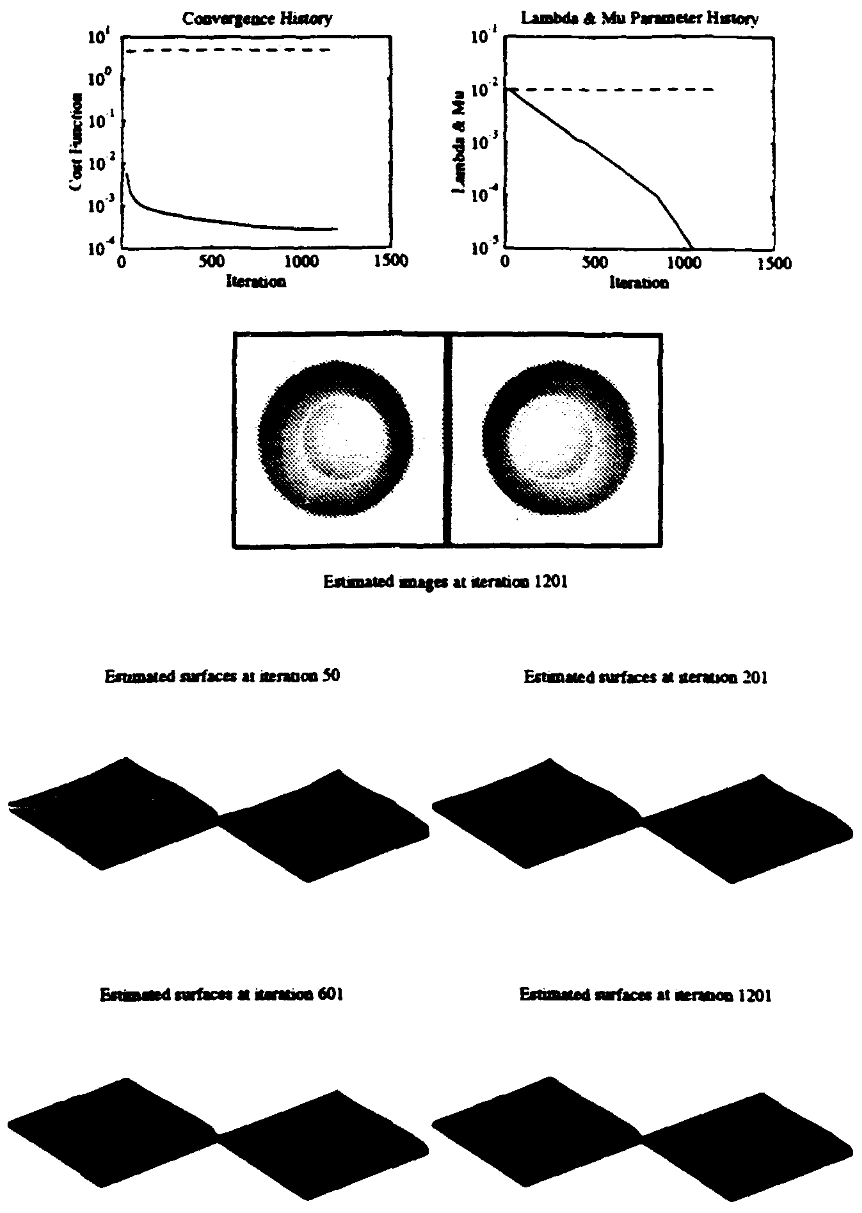


Figure 5-19: Performance of the dual-z algorithm on the hard crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ (solid line) and μ (dashed-line) cost function parameters. Also shown are images based on the estimated surfaces and mesh plots of the surfaces at different points during optimization.

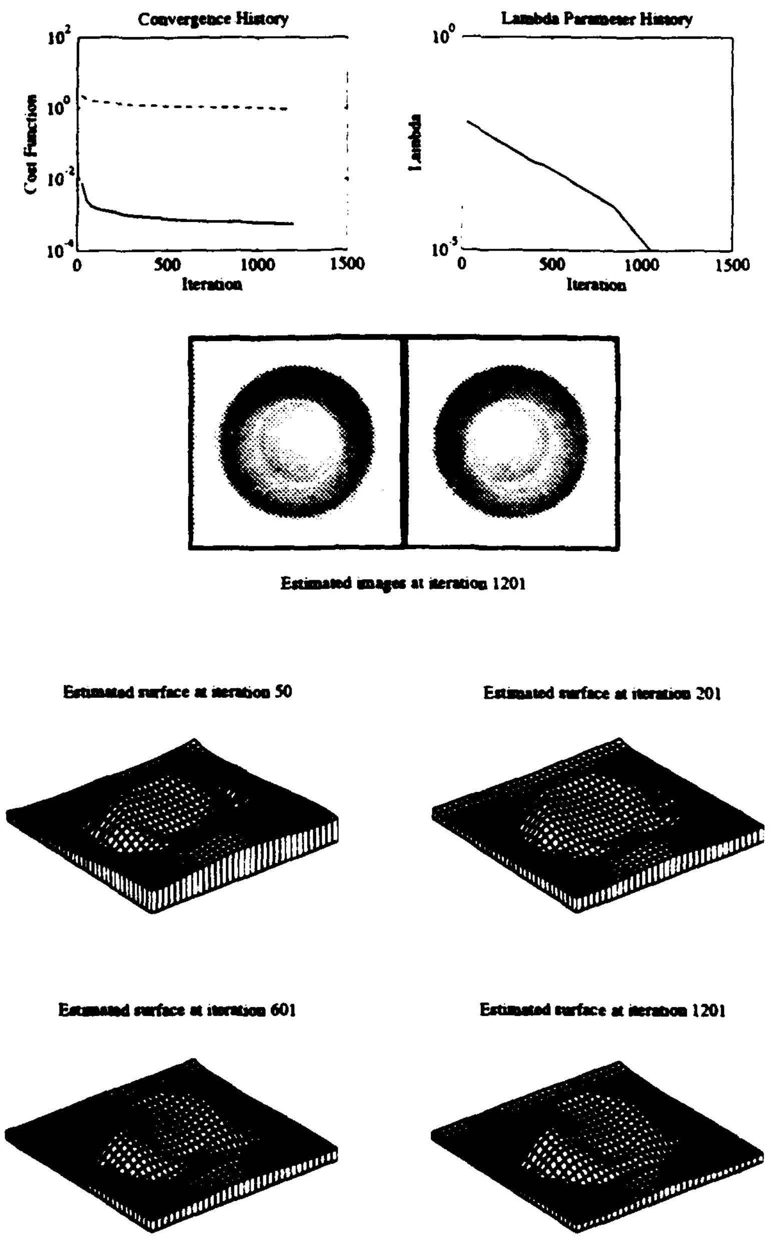


Figure 5-20: Performance of the disparity-based algorithm on the hard crater images. The upper-left graph shows the cost function (solid line) and RMS error of the estimated surface (dashed-line) plotted against the number of function evaluations. The upper-right graph shows the history of the λ cost function parameter. Also shown are images based on the estimated surface and mesh plots of the surface at different points during optimization.

Absolute Error

Algorithm	Easy Crater	Hill	Mountain	Hard Crater
dual-z	1.2991	0.2274	0.8774	2.4241
zpq	0.2130	0.0247	2.7494	1.5933
disparity	0.1668	0.0251	0.8383	0.9241
z-only	0.1728	0.0247	0.7799	1.0716

Relative Error

Algorithm	Easy Crater	Hill	Mountain	Hard Crater
dual-z	0.3614	0.0968	0.8436	1.0078
zpq	0.1678	0.0205	2.5715	1.5379
disparity	0.1663	0.0176	0.7193	0.9240
z-only	0.1721	0.0175	0.7177	1.0078

Table 5-2: Absolute and relative surface estimation error by algorithm and test case (1200 iterations).

5.5 Summary

A summary of running the algorithms on the test images is shown in Table 5-2 and Figure 5-21. The table shows the relative and absolute error between the true and estimated surface at the last iteration. The absolute error is computed using the formula

$$J_{\text{abs}} = \sqrt{\frac{1}{NM}\sum_{x,y}(z - z^*)^2}, \tag{5.1}$$

while the relative error is computed using

$$J_{\text{rel}} = \sqrt{\frac{1}{NM}\sum_{x,y}((z - \bar{z}) - (z^* - \bar{z}^*))^2}. \tag{5.2}$$

where z^* is the true surface height and $\bar{z}$ is the average of z. The figure shows the absolute error normalized by the surface depth change of the true surface (i.e., $(z_{max} - z_{min})$). The figure can be used to interpret the error as a fraction of the total surface depth change. All the algorithms were run with 6 hierarchical basis levels which is one less than the largest number of levels that can be used with 65-by-65 images. Using this number of hierarchical bases sped up the convergence of each algorithm by a factor of 3–5.

It is clear that all of the algorithms had problems with the hard crater images. Each one got caught in the local minimum. The reason for this can be seen in the reflectance contours of Figure 5-16. For a given brightness level (i.e., along one of the contours), there are two viable solutions with different surface orientations. The

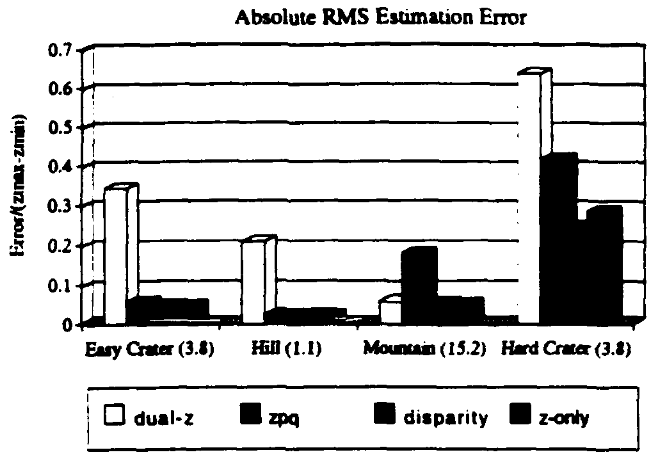


Figure 5-21: Absolute surface estimation error scaled by the true surface depth change. The depth change is shown in parentheses.

local minimum has a orientation in the "dipped" region that is a viable but incorrect orientation, and the stereo information is not strong enough to pull it out of the local minimum.

The hard crater images have been the driving force behind much of this research and are one of the reasons that so many alternate algorithms were implemented. As each algorithm was created and implemented it was hoped that the new search space would allow the correct interpretation of the hard crater images. Alas, every algorithm I have tried has been caught in the local minimum.

The only way the algorithms can correctly interpret the hard crater images is to perform the optimization with no hierarchical basis levels. The convergence is very slow but the solution is correct. I hesitate to recommend that all runs be performed using the nodal basis, however, since the performance hit is huge and there is no guarantee that the true solution will be found. Consider the fact that the test images are small and that more normal size images are of the order 513-by-513. One function evaluation with these images will take 64 times as long to compute as the test images. In addition, since all of the algorithms are diffusion-type algorithms, we could expect an 8 times increase in the number of function evaluations to obtain convergence if the nodal basis is used. As you can see, the ramifications of not using the hierarchical bases are huge.

Fortunately, the fact that the algorithms are caught in a local minimum is easy to see for the hard crater images, since the estimated images that are created as a by-product of each function evaluation show streaks where the photo-topography constraints are not met (these streaks are difficult to see in the halftoned reproductions of the gray level images). Considering this fact, I recommend that the largest number of hierarchical basis levels be used to speed convergence and the results checked by looking at the estimated images.

Clearly, the algorithm that performed best is the z-only algorithm. Not only is it the quickest algorithm to compute (i.e., one iteration of the z-only algorithm computes faster that one iteration of any of the other algorithms),[5] but it also has the best convergence rate. This algorithm and the next best algorithm, the disparity algorithm, were created by directly implementing the problem constraints as a cost function. There might be a lesson here that the best algorithms are formed by incorporating the constraint equations into the cost function in the most direct and straightforward way.

[5]The evaluation time for the algorithms is approximately proportional to the number of free variables: N^2 for z-only and disparity, $3N^2$ for zpq, and $2N^2$ for dual-z.

Chapter 6

Comparison with Shape-from-Shading

In this chapter I show how a simple shape-from-shading algorithm performs on my test images. By restricting the z-only algorithm to work on a single image and assuming orthographic projection I can create an algorithm similar to Szeliski [Szeliski, 1991] and Leclerc and Bobick [Leclerc and Bobick, 1991]. The resulting algorithm is based on the cost function,

$$\min_z J = \frac{1}{2} \iint \left\{ (E(x,y) - R(p,q))^2 + \lambda(\nabla^2 z)^2 \right\} dx\, dy \tag{6.1}$$

where

$$\begin{aligned} p &= z_x, \\ q &= z_y. \end{aligned} \tag{6.2}$$

The shape-from-shading algorithm is based on z but cannot estimate the true depth: the algorithm can only estimate the shape. The depth bias is unobservable to this algorithm.

Following the development of the z-only algorithm, the discretized version of this cost function is

$$\min_z J = \frac{1}{2NM\epsilon^2} \sum_{x,y \in \mathcal{D}} \left\{ (E(x,y) - R(p,q))^2 + \lambda(\nabla^2 z)^2 \right\}. \tag{6.3}$$

Note that this cost function directly uses the values in $E(x,y)$ without having to interpolate as was required when forming $F(x,y)$ for the z-only algorithm.

I show the performance of the shape-from-shading algorithm on the left image of four test image sets in Figures 6-4–6-3. The figures show the cost function history anu optimization parameters as a function of the number of function evaluations. Also shown is the estimated surface shape at various stages during the convergence.

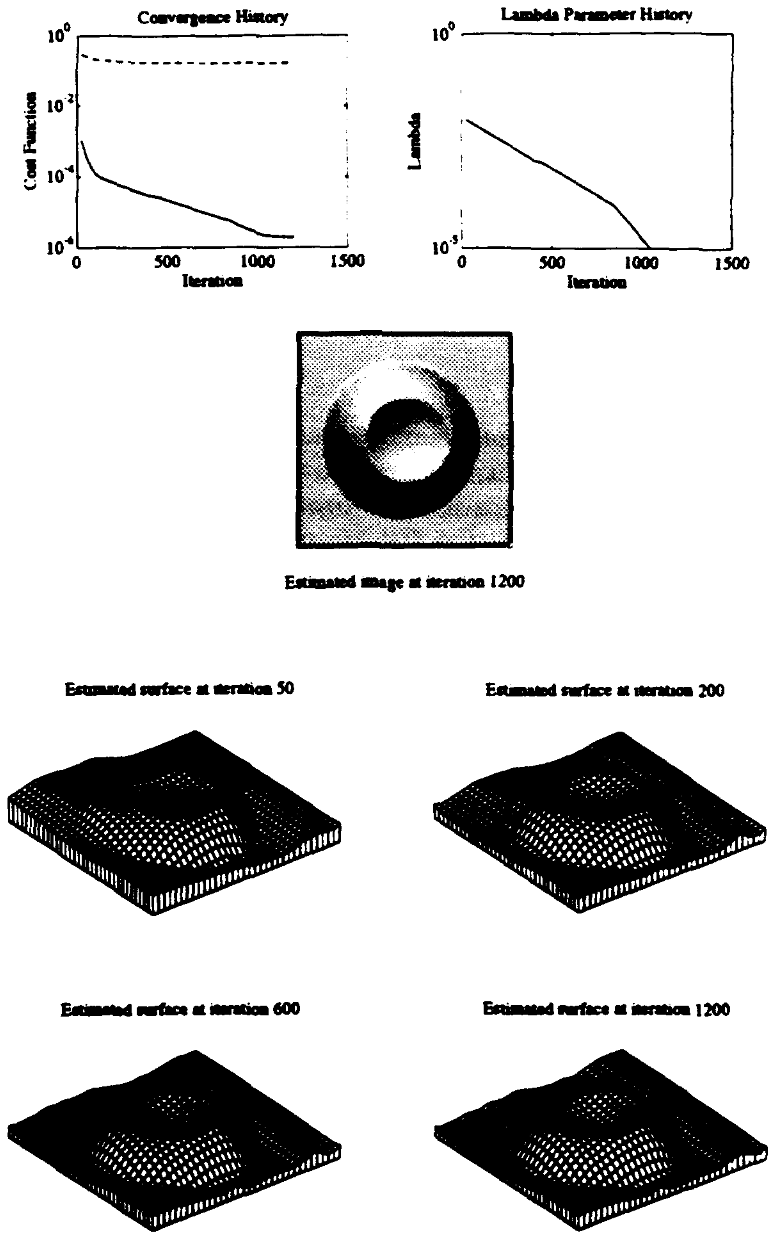


Figure 6-1: Performance of the shape-from-shading algorithm on the left easy crater image.

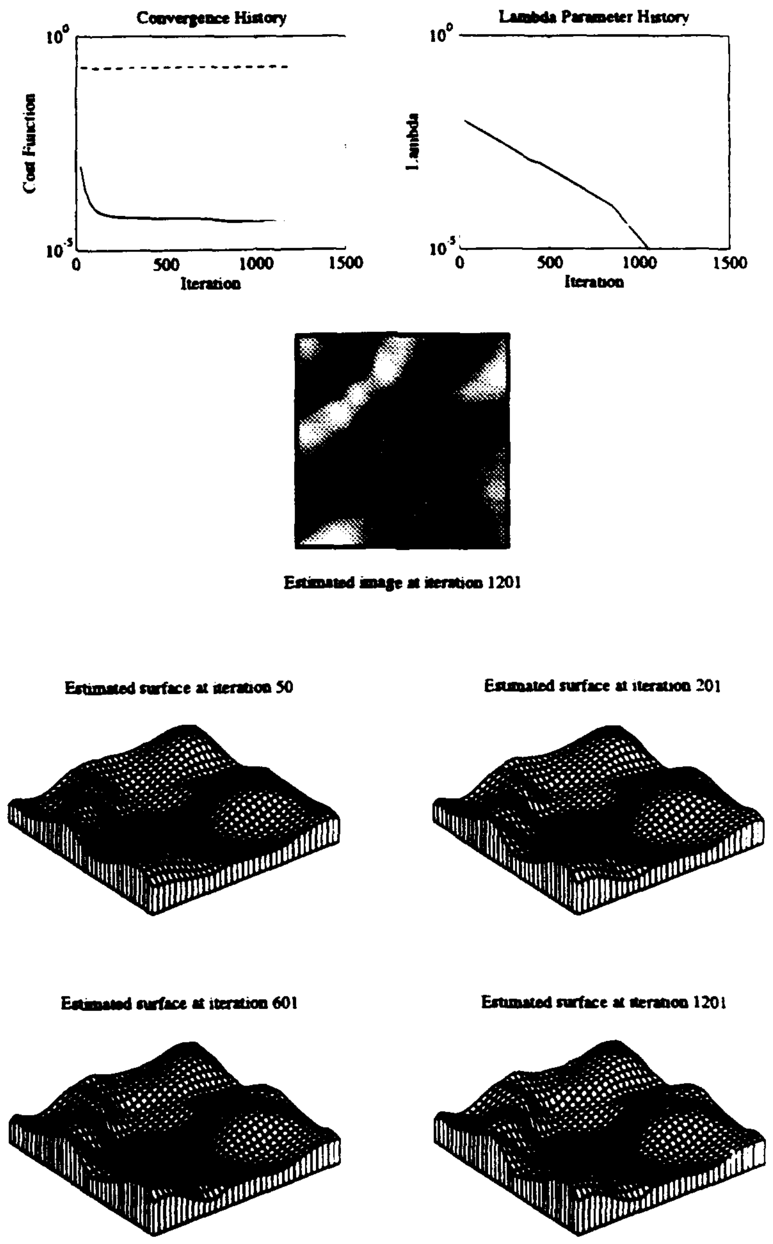


Figure 6-2: Performance of the shape-from-shading algorithm on the left hill image.

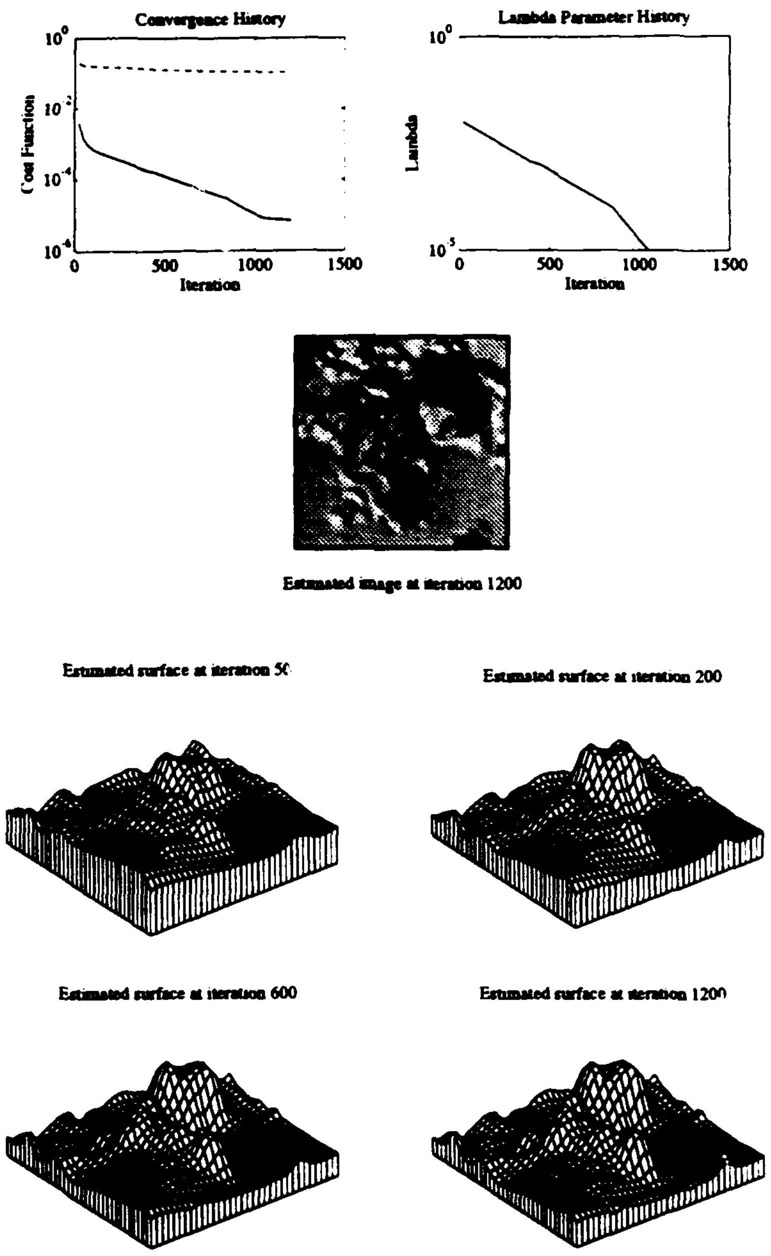


Figure 6-3: Performance of the shape-from-shading algorithm on the left mountain image.

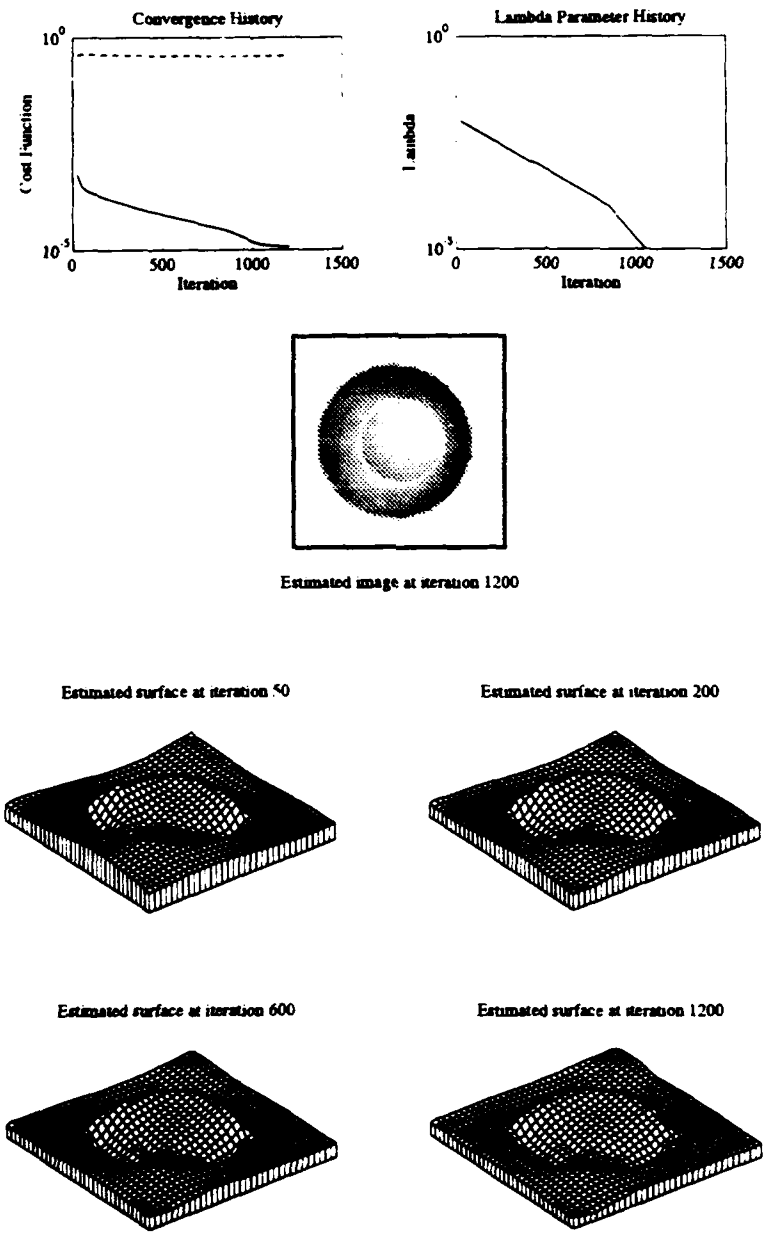


Figure 6-4: Performance of the shape-from-shading algorithm on the left hard crater image.

Image set	Rel. SFS Error	Rel. z-only Error
Easy Crater	0.6242	0.1721
Hill	0.2083	0.0175
Mountain	1.6343	0.7177
Hard Crater	1.3532	1.0715

Table 6-1: Estimation error of SFS algorithm on the test images compared to z-only results (1200 iterations).

The figures show that the shape-from-shading algorithm can determine the shape of the surface, but with some error. I have summarized the final error between the estimated shape and true shape in Table 6-1. The estimated surface and true surface offset so that their average heights coincide before computing the errors shown in the table. The error was computed using Equation 5.2.

The estimation error when using shape-from-shading alone is greater than the estimation error when using the z-only algorithm. While this is not surprising since the z-only algorithm has two images to work with instead of one, it is nice to see that significant performance gains are possible when using a fused algorithm.

Chapter 7

Error Analysis

It is equally important when developing algorithms to investigate their robustness in the face of errors. In this chapter I present the results of running the z-only algorithm on hill images that contain errors. I have introduced errors of three types:

1. measurement errors,
2. geometry errors,
3. reflectance errors.

Each error is introduced separately to simplify the analysis. In each case, a figure is presented that shows the relative and absolute error between the true and estimated surface, along with the estimated surface, convergence history and estimated images for the worse case. All the results below are the error after 600 function evaluations (about 300 updates). The initial condition for each case was a flat plane at the nominal depth. For reference, the relative and absolute errors after 600 function evaluations on noise-free images are 0.0175 and 0.0247 respectively.

7.1 Measurement Errors

To determine the performance of the z-only algorithm on noisy images I added Gaussian white noise (with signal-to-noise ratios (SNR) of 100, 10, or 2) to the hill images. The resulting brightness values were then clipped to the range $[0, 1]$. The results of running the z-only algorithm on these noisy images are shown in Figure 7-1 and Table 7-1. The figure shows that the algorithm performs well even for a signal-to-noise ratios of 2 and that the estimate degrades gracefully as the SNR is decreased.

7.2 Camera Geometry Error

The effect of errors in the camera geometry were investigated by adding error to the baseline distance, interior orientation parameters, or by rotating the images to move

Case	Change	Rel. Error	Abs. Error
	Nominal (z-only)	0.0175	0.0247
1	SNR 100	0.0203	0.0275
2	SNR 10	0.0751	0.1323
3	SNR 2	0.1436	0.1701

Table 7-1: RMS Estimated surface error from noisy images.

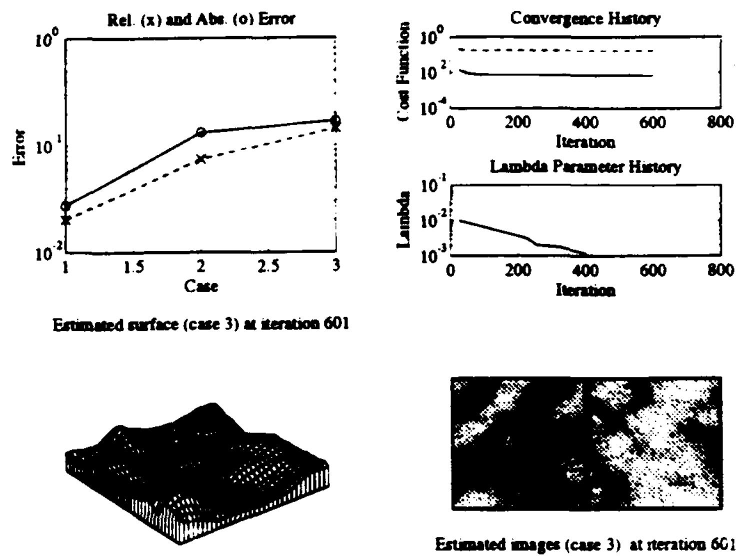


Figure 7-1: Performance of z-only algorithm on noisy hill images

the epipolar lines out of alignment. Each change was introduced independently. The baseline error images were created by changing both the baseline and nominal depth by the same percentage in order to keep the interior orientation parameters constant (10%, 20%, and 30% changes where introduced). The interior orientation error images were created by offsetting the origin of the camera coordinates in a random direction with magnitude of 1, 3, or 6 pixels. The epipolar error images were created by rotating the images so the total error introduced between the images was 0.5, 1, or 3 degrees.

The results of running the z-only algorithm on these changed images are shown in Figures 7-2–7-4 and Table 7-2. The baseline errors mostly affect the absolute depth of surface estimate and have little effect on the relative error of the surface. On the other hand, the small change I introduced into the interior orientation parameters results in a profound change in the estimated surface (see Figure 7-3). Clearly it is important to have accurate knowledge of the camera parameters to obtain good

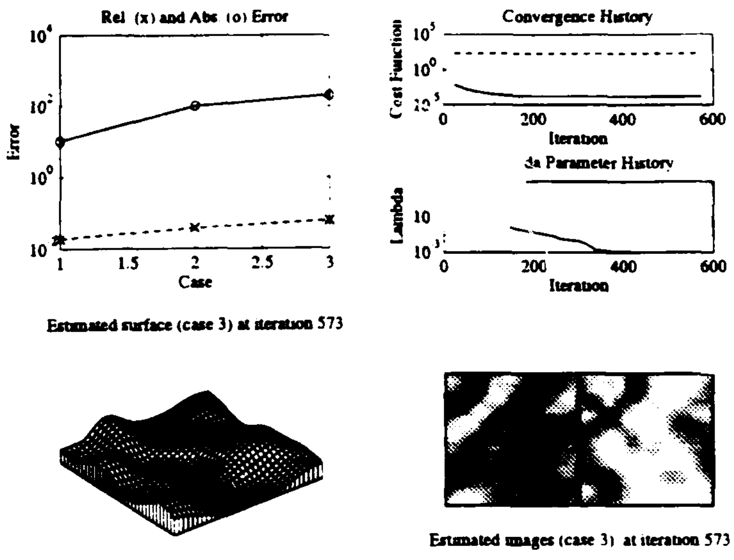


Figure 7-2: Performance of z-only algorithm with hill baseline errors.

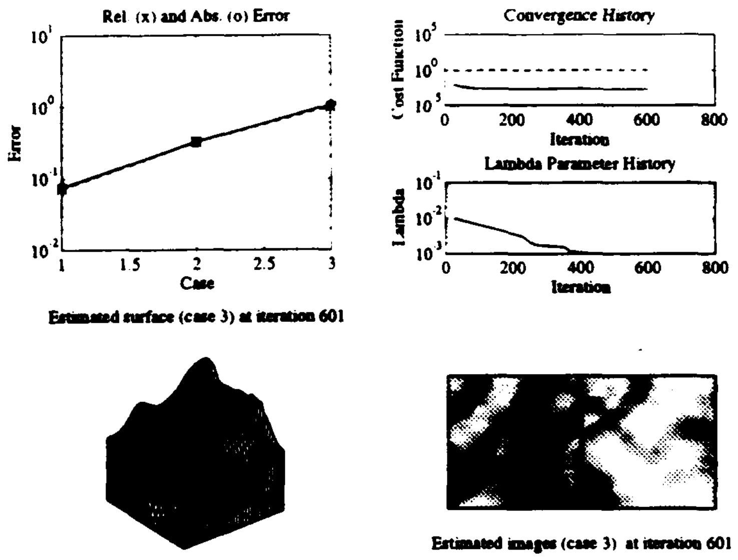


Figure 7-3: Performance of z-only algorithm with hill interior orientation errors.

Case	Change	Rel. Error	Abs. Error
	Nominal (z-only)	0.0175	0.0247
1	baseline error (−10%)	0.0189	10.0075
2	baseline error (−20%)	0.0374	99.9331
3	baseline error (−30%)	0.0609	199.8330
1	interior orientation error (1 pixel)	0.0716	0.0744
2	interior orientation error (3 pixels)	0.3214	0.3228
3	interior orientation error (6 pixels)	0.9985	1.0506
1	epipolar error (0.5°)	0.0319	0.0499
2	epipolar error (1°)	0.0570	0.0630
3	epipolar error (3°)	0.0943	0.2432

Table 7-2: RMS Estimated surface error from geometry errors.

estimates.

Figure 7-4 shows that a moderate amount of error in the epipolar calibration can be tolerated with only a small effect on the estimated surface. This is good news since most planetary images will have to be re-projected into the aligned optical axes coordinate system. Figure 7-4 shows that the algorithm can tolerate some errors in this projection and still produce meaningful results.

7.3 Reflectance map errors

The effects of errors in the reflectance map were investigated by changing the light source positions or by scaling brightness values in the images to simulate an error in albedo calibration. I did not investigate the effects of using the wrong reflectance (such as Minnaert reflectance) since the estimation errors due to this effect cannot be generalized. The light source error images were created by changing the light source direction by 5, 15, or 30 degrees in a random direction. The albedo error images were created by scaling the images by 1/0.99, 1/0.95, or 1/0.90. The results of running the z-only algorithm on these changed images are shown in Figures 7-5–7-6 and Table 7-3 Figure 7-5 shows that light source position errors of up to 30 degrees have relatively little effect on the estimated surface. While, typical photo-topography images have light source (sun) positions that are known to high precision, these results show that the z-only algorithm could be applied to images where the light sources are not as well known.

On the other hand, Figure 7-6 shows that the algorithm can tolerate albedo calibration errors up to 10%. These results are misleading, however, since the algorithms fail to converge when the albedo error is greater than 10%. The algorithm may not

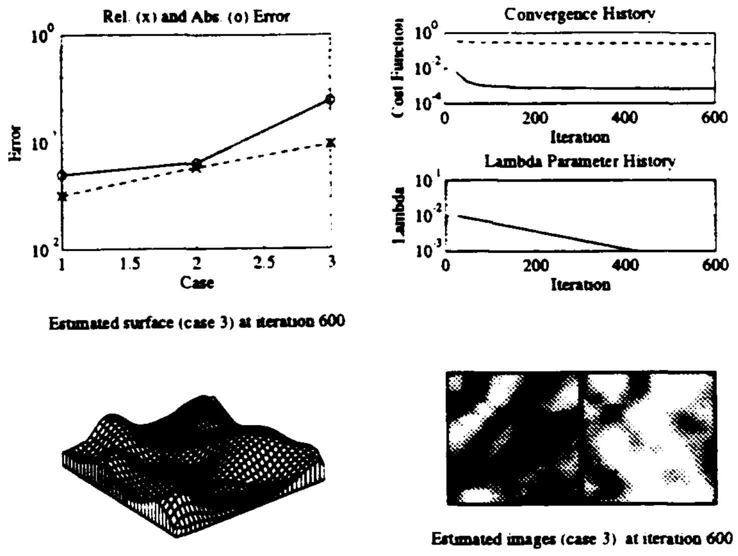


Figure 7-4: Performance of z-only algorithm with hill epipolar errors.

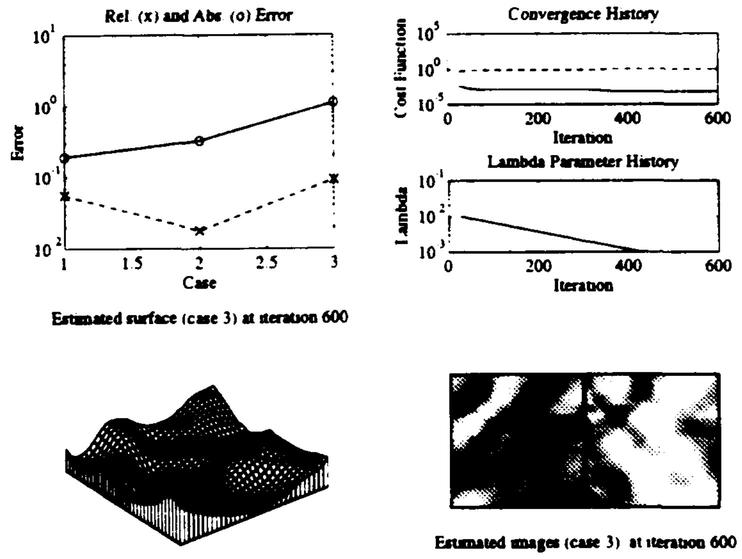


Figure 7-5: Performance of z-only algorithm with hill light source errors.

Case	Change	Rel. Error	Abs. Error
	Nominal (z-only)	0.0175	0.0247
1	light source error (5°)	0.0551	0.1912
2	light source error (15°)	0.0178	0.3219
3	light source error (30°)	0.0922	1.1150
1	albedo error (1%)	0.0241	0.0341
2	albedo error (5%)	0.0681	0.0686
3	albedo error (10%)	0.1234	0.1253

Table 7-3: RMS Estimated surface error from reflectance errors.

converge in those cases since the Lambertian reflectance map cannot exceed 1.0 and the error images produced contain many brightness values beyond that limit. The algorithm can probably tolerate albedo calibration errors that don't create normalized brightness values greater than 1.0.

7.4 Summary

The foregoing plots and tables indicate the z-only algorithm is fairly robust. In particular, they indicate that the algorithm can produce a reasonably good estimate even if the images are noisy, the camera geometry is not known perfectly, and the reflectance properties are in error. They also show that it is very important to have accurate internal orientation parameters, and an accurate baseline in order to estimate the depth correctly.

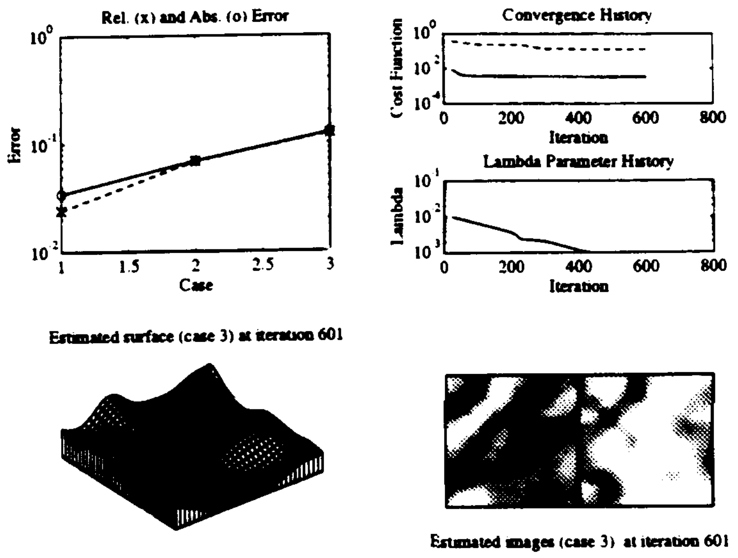


Figure 7-6: Performance of z-only algorithm with hill albedo errors.

Chapter 8

Algorithm Extensions

In this chapter I discuss two ways of extending the algorithm to varying albedo surfaces. While all the previous algorithms have been restricted to surfaces with a constant (known) albedo, the algorithms discussed in this chapter work for surfaces that have markings or striations. The new algorithms still require that the geometric reflectance properties be constant and known for the whole surface.

I also show in this chapter how more general camera geometries can be accommodated. Basically, the images are projected into a coordinate system that has aligned optical axes.

8.1 Varying albedo algorithms

As mentioned in Section 2.5, the simplification of constant albedo severely restricts the applicability of the algorithms that are developed in Chapter 4. In this section, I lift that restriction. The algorithms that result, do converge to a solution close to the actual surface, but the convergence is slower than for the constant albedo algorithms.

The first thing to understand is whether we would expect a varying albedo algorithm to work. In other words, 'Does it seem reasonable that a unique value of albedo can be chosen for each point in the image'? Let's investigate that question.

Each point in the images provides two constraints via the two *extended photo-topography equations*

$$\begin{aligned} E^{(1)}(x + \frac{fb}{2z}, y) &= \rho(x,y)R^{(1)}(p,q), \\ E^{(2)}(x - \frac{fb}{2z}, y) &= \rho(x,y)R^{(2)}(p,q), \end{aligned} \tag{8.1}$$

for the two unknowns z and ρ. Since this is a situation with two equations and unknowns, a solution is at least conceivable. Investigating further we find that for a given $\rho(x,y)$ there are at most two gradient directions $(p(x,y), q(x,y))$ that can satisfy each equation.[1] These gradient directions cannot be chosen arbitrarily since they must be consistent with the underlying height z, via the integrability constraint equations.

$$\begin{aligned} p &= \frac{f z_x}{x z_x + z}, \\ q &= \frac{f z_y}{y z_y + z}. \end{aligned} \tag{8.2}$$

A solution is obtained when a common, consistent gradient estimates exist. Conversely, a given z defines the gradient components and the relationship between points in the images. A solution is obtained when ρ can be chosen to match the images $E^{(i)}$ with the images estimated from the reflectance maps.

From the foregoing discussion it should be clear that it is possible to create image sets that are inconsistent so that no solution exists. On the other hand it should also be clear that given consistent images a solution exists. Thus we find that the extended problem is well-posed.

8.2 Minimizing departure from a constant albedo.

The first algorithm is applicable to slowly varying albedo surfaces. It is based on the z-only cost function but includes a penalty term of the form $\mu(\rho - \bar{\rho})^2$ where $\bar{\rho}$ is the average albedo over the whole image. This additional term penalizes departure from a constant albedo and can be used to estimate the calibration factor $\lambda\rho$ on the reflectance. The cost function including this term is

$$\begin{aligned} \min_z J = \frac{1}{2} \iint \Bigg\{ & \left(E^{(1)}(x + \frac{fb}{2z}, y) - \rho(x,y) R^{(1)}(p,q) \right)^2 \\ & + \left(E^{(2)}(x - \frac{fb}{2z}, y) - \rho(x,y) R^{(2)}(p,q) \right)^2 \\ & + \lambda \left[z_{xx}^2 + 2 z_{xy}^2 + z_{yy}^2 \right] + \mu(\rho - \bar{\rho})^2 \Bigg\} \, dx\, dy \end{aligned} \tag{8.3}$$

where

$$\bar{\rho} = \frac{\int_{\mathcal{D}} \rho \, dx\, dy}{\int_{\mathcal{D}} dx\, dy} \tag{8.4}$$

and $\mathcal{D}$ is the whole image. The smoothness penalty term is used to guide convergence and is not required to guarantee a unique solution. Usually, λ is slowly reduced to zero as the solution is reached to avoid biasing the solution. The algorithm based on this cost function converges, albeit slowly, and works best for surfaces that have nearly constant albedo.

[1] For surfaces with Lambertian Reflectance

8.3 Minimizing local albedo change.

The second algorithm is applicable to surfaces with piecewise constant or piecewise linear albedo. In this case, the cost function includes a penalty term of the form $\mu(\rho - \bar{\rho})^2$ where $\bar{\rho}$ is the average in some local neighborhood $\mathcal{N}$. This term is similar to a discrete approximation to the Laplacian and penalizes departure from a local average. For a piecewise constant albedo, this term will be zero except on the boundary between the constant areas of albedo. The cost function including this term is

$$\begin{aligned}\min_z J = \frac{1}{2}\iint\Big\{&\left(E^{(1)}(x + \frac{fb}{2z}, y) - \rho(x,y)R^{(1)}(p,q)\right)^2 \\ &+ \left(E^{(2)}(x - \frac{fb}{2z}, y) - \rho(x,y)R^{(2)}(p,q)\right)^2 \\ &+ \lambda\left[z_{xx}^2 + 2z_{xy}^2 + z_{yy}^2\right] + \mu(\rho - \bar{\rho})^2\Big\}\,dx\,dy \qquad (8.5)\end{aligned}$$

where

$$\bar{\rho} = \frac{\int_{\mathcal{N}} \rho dx\,dy}{\int_{\mathcal{N}} dx\ dy}. \qquad (8.6)$$

Following the implementation of the z-only algorithm, the cost function can be discretized to become.

$$\begin{aligned}\min_z J = \frac{1}{2NM\epsilon^2}\sum_{x,y\in\mathcal{D}}\Big\{&\left(F^{(1)}(x,y) - \rho(x,y)R^{(1)}(p,q)\right)^2 \\ &+ \left(F^{(2)}(x,y) - \rho(x,y)R^{(2)}(p,q)\right)^2 \\ &+ \lambda\left[z_{xx}^2 + 2z_{xy}^2 + z_{yy}^2\right] + \mu(\rho - \bar{\rho})^2\Big\} \qquad (8.7)\end{aligned}$$

where $\bar{\rho}$ is the average in a 3-by-3 neighborhood,

$$\bar{\rho} = \frac{1}{9}\begin{array}{ccc}\boxed{1} & \boxed{1} & \boxed{1} \\ \boxed{1} & \boxed{1} & \boxed{1} \\ \boxed{1} & \boxed{1} & \boxed{1}\end{array} \diamond \rho. \qquad (8.8)$$

The performance of this algorithm on the varying albedo test images is shown in Figures 8-2–8-4.

8.4 Varying albedo test results.

To test the algorithm based on this cost function, two of the test images were modified to include either a $\rho = 0.7$ albedo strip from the lower left to the upper right (Figure 8-1) or an $\rho = 0.7$ albedo variation in a layer similar to what would be expected from sedimentation processes (Figure 8-3).

8.4.1 Crater with dark stripe

The first set of test images has the $\rho = 0.7$ albedo stripe imposed on the easy crater test case. The light source geometry, true surface, and true albedo are shown in Figure 8-1. The albedo variation is "painted" on the crater surface.

The Figure 8-2 shows the result of applying the varying albedo algorithm to this test case. The figure shows the cost function history and optimization parameters as a function of the number of function evaluations. Also shown are the estimated surface and albedo image at various stages during the convergence. While the algorithm does converge to a surface and albedo map close to the true values, the convergence is much slower than for the constant albedo algorithm and the final surface estimate has more error. Of course, the constant albedo algorithm would not perform any better if presented with the varying albedo images.

8.4.2 Hill with sedimentation

The other set of varying albedo test images is based on the *hill* test case with the albedo set to $\rho = 0.7$ for surface heights in a certain range (see Figure 8-3). The light source positions and geometry are the same as for the hill test case.

Figure 8-4 shows the result of applying the varying albedo algorithm to this test case. The figure shows the cost function history and optimization parameters as a function of the number of function evaluations. Also shown is the estimated surface and albedo image at various stages during the convergence. While the algorithm does converge to a surface and albedo map close to the true values, the convergence is slower than for the constant albedo algorithm and the final surface estimate has more error.

8.4.3 Summary

The varying albedo algorithm can successfully estimate both the surface depth and the albedo variation but requires at least 2 times as many function evaluations than the constant albedo z-only algorithm. In fact, the performance figures in this section show surfaces that have not converged even after 2800 function evaluations. However, these algorithms must be used for surfaces that cannot be estimated correctly with the z-only algorithm because of varying albedo.

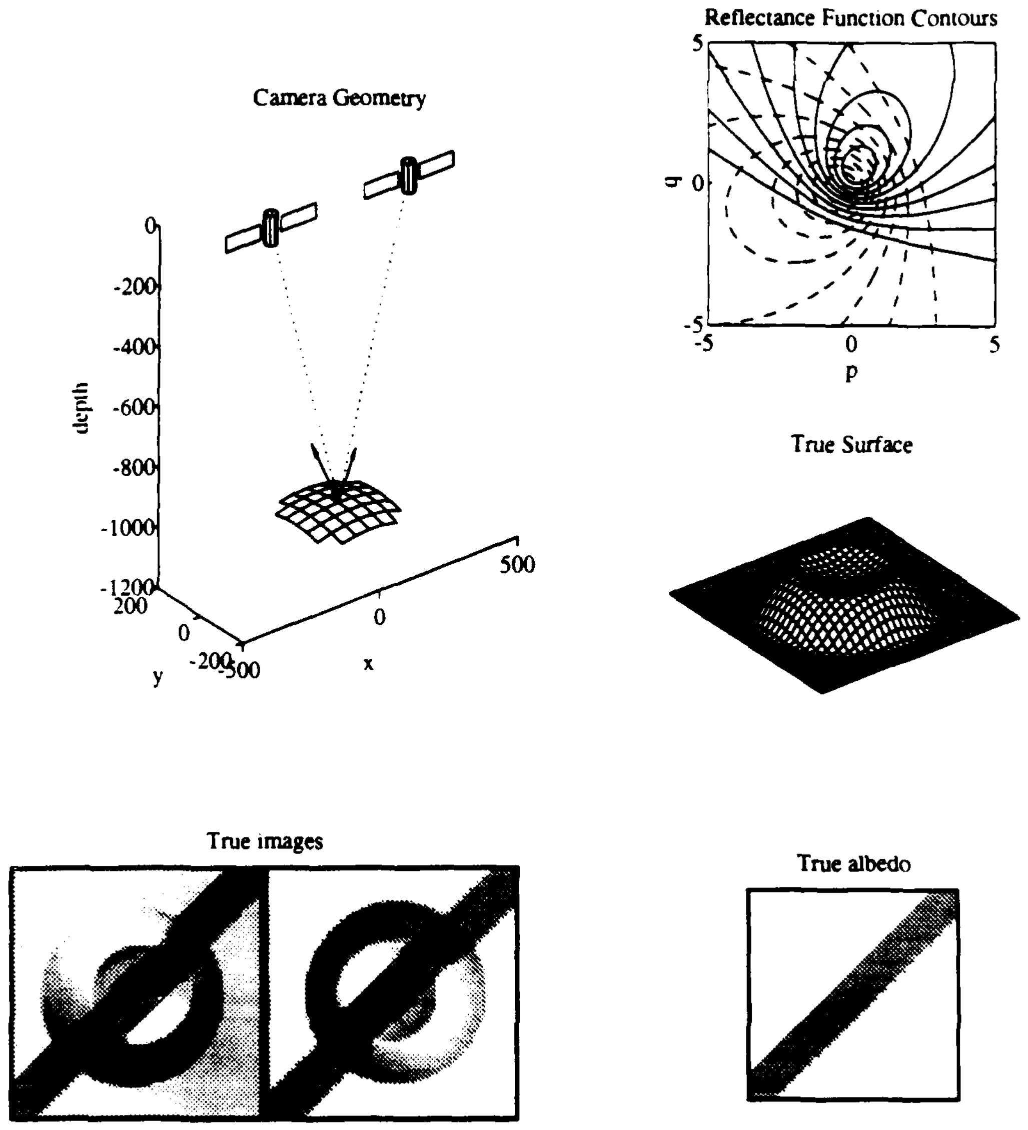


Figure 8-1: Crater on flat plane with varying albedo. Shown is the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, albedo image, and reflectance function contours for the two light source positions.

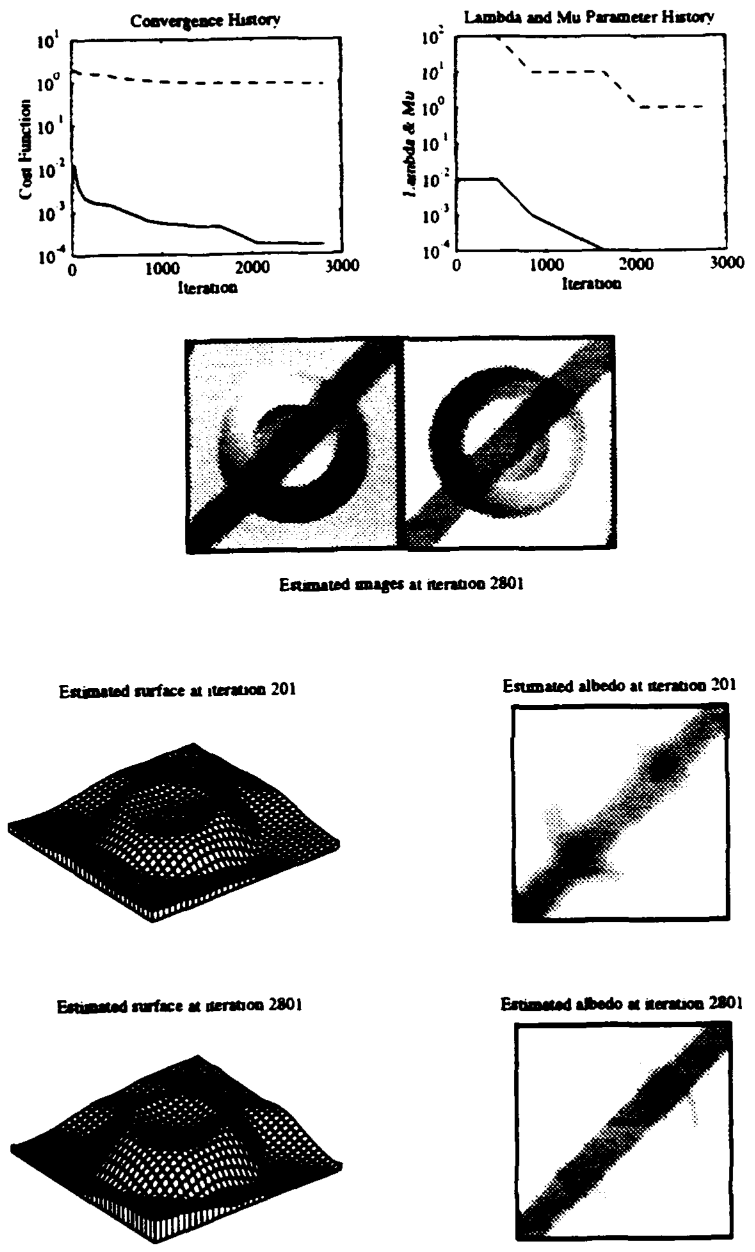


Figure 8-2: Performance of the varying albedo algorithm on the varying albedo crater images.

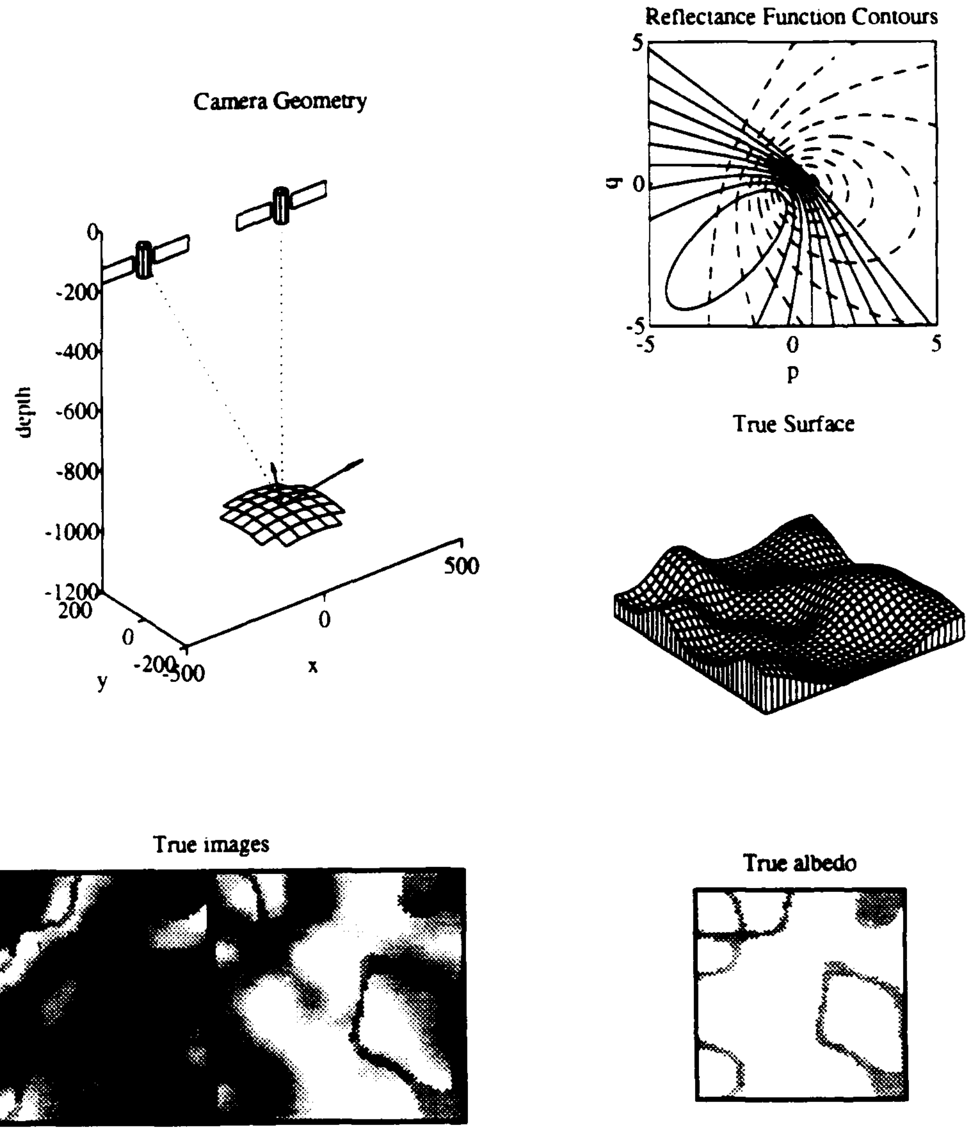


Figure 8-3: Test images of hill with varying albedo. Shown is the camera geometry as projected into the xz- and yz-planes, the true surface as a mesh plot, albedo image, and reflectance function contours for the two light source positions.

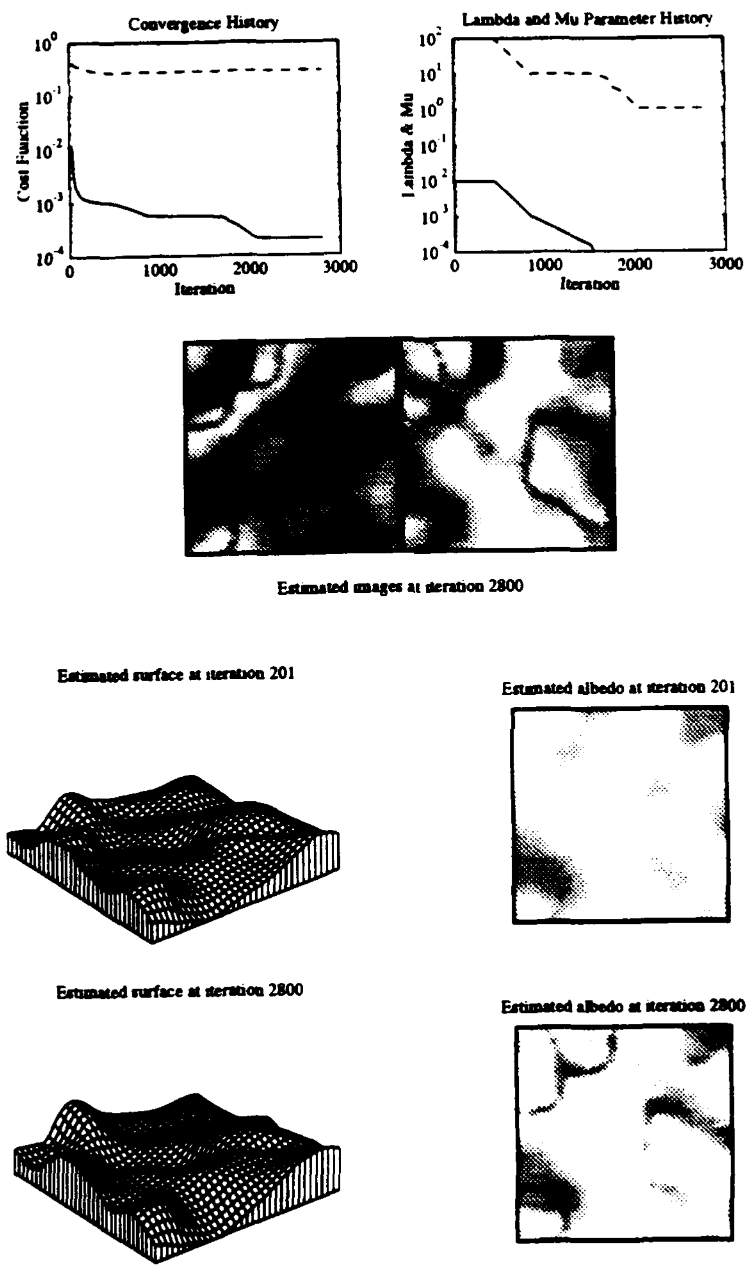


Figure 8-4: Performance of the varying albedo algorithm on the varying albedo hill images.

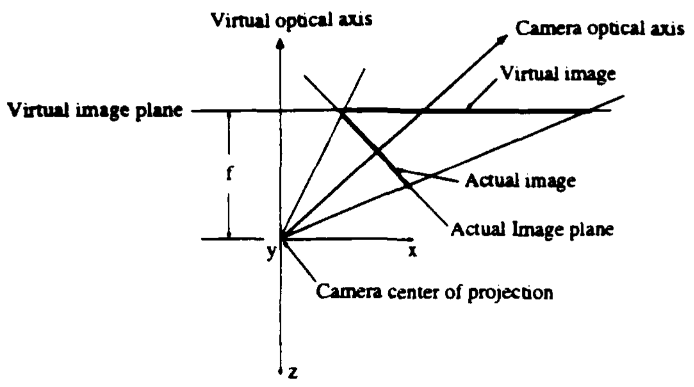


Figure 8-5: Camera reprojection geometry.

The slow convergence is a direct result of increasing the number of degrees of freedom available to the algorithm. It has the choice of meeting the photo-topography constraints by changing the depth or by just changing the albedo. Initially it's easier to change the albedo since it shows up directly in the constraint equations. However, using albedo changes alone is not sufficient to minimize the equations, so after a while the surface height begins the change.

The convergence would probably be faster if the stereo part of the algorithm was more powerful. The disparity of the albedo edges should provide a strong constraint on the height. The fact that they don't is due to the weakness of the stereo part of the algorithm. Hopefully ongoing research will turn up a more powerful stereo algorithm.

8.5 General Camera Geometry

The algorithms presented thus far are based on the simplification that the camera optical axes are aligned. This is never true for images taken from moving vehicles such as inner-planetary probes, satellites, or aircraft. The simplification of aligned optical axes is used since it greatly simplified the equations. The simplification can be lifted by re-projecting the images from their true camera coordinates into the coordinates of a virtual camera with its principal point in the same position but with the aligned orientation (see Figure 8-5). I refer to the true camera coordinates as the primed coordinate system (i.e., $(x', y', z')^T$ and the aligned coordinates using un-primed notation, consistent with the equations presented so far.

Points in a camera image map to *rays* in space. The mapping between coordinate systems preserves the orientation of these rays in the global coordinate system. Let $\mathbf{r}' = (x', y', f')^T$ be the coordinates of a point in the camera's image. This maps to the ray in the direction $\mathbf{r}'$. Let T be the rotation matrix relating the two coordinate systems, that is,

$$\mathbf{r}' = T\mathbf{r}, \tag{8.9}$$

for any vector $\mathbf{r}$ (see Figure 8-5). The ray in the aligned coordinate system is then along the direction $T\mathbf{r}'$. This ray can be re-projected into the virtual image plane with focal length f by normalizing,

$$\mathbf{r} = \frac{f(T^T\mathbf{r}')}{(T^T\mathbf{r}') \cdot \hat{\mathbf{z}}} \tag{8.10}$$

where $\hat{\mathbf{z}} = (0, 0, 1)^T$ is the unit vector along the z axis in the aligned coordinate system. Similarly,

$$\mathbf{r}' = \frac{f(T\mathbf{r})}{(T\mathbf{r}) \cdot \hat{\mathbf{z}}'} \tag{8.11}$$

relates points in the un-primed image to points in the primed image.

We can use these equations to create the image that would have been seen by a camera with aligned optical axes. Given $E'(r')$ in the true camera coordinate system. the re-projected image is

$$E(\mathbf{r}) = E'\left(\frac{f(T\mathbf{r})}{(T\mathbf{r}) \cdot \hat{\mathbf{z}}'}\right) \tag{8.12}$$

since T^T is the rotation matrix from the aligned coordinates to the true coordinates. For $\mathbf{r} = (x, y, f)^T$ this equation written in (x, y) notation is

$$E(x, y) = E'(\frac{t_{11}x + t_{12}y + t_{13}f}{t_{31}x + t_{32}y + t_{33}f}, \frac{t_{21}x + t_{22}y + t_{23}f}{t_{31}x + T_{32}y + t_{33}f}). \tag{8.13}$$

where the elements of the 3-by-3 matrix T are t_{ij}. For discrete images, the value of the above expression can be computed using some type of interpolation (say bilinear interpolation).

Note that this re-projection can be easily incorporated into the z-only, disparity-based, and varying albedo algorithms by defining $F^{(i)}$ to interpolate in E' instead of E.

Chapter 9

Summary

This thesis has presented a methodology for combining or fusing multiple vision algorithms. Four different cost functions (and their associated algorithms) were presented that illustrate the methodology. The basic methodology is to combine the constraint equations of the problem to form a single cost function in the spirit of variational calculus.

The performance of the four algorithms was evaluated using four synthetic noise-free test images of varying difficulty. The most closely-coupled algorithm, the z-only algorithm, had the best performance. The z-only algorithm was able to correctly estimate the synthetic surface in about 200 function evaluations for three of the four cases. For the remaining case (the hard crater images), the algorithm got stuck in a local minimum, as did all the algorithms tried. This case has lighting geometry that results in ambiguous shading information.

It was shown that the z-only algorithm has much better performance using the two photo-topography images than a simple shape-from-shading algorithm which uses only one image. This performance increase validates the fusion approach to obtaining better performing vision algorithms.

The z-only algorithm was also shown to be robust; able to accurately estimate the synthetic surface in the presence of several types of errors. The performance of the algorithm based on images that contained noise, geometry error, or reflectance errors was shown. In most cases, the algorithm was able to form a good estimate.

9.1 Mars Images

The robustness and performance of the algorithm on synthetic images, builds confidence that the algorithm can perform similarly on real images. One such set of real images is shown in Figure 9-1.[1] The images are Viking stereo images of Mars. The images as received (and shown) are processed versions of the original Viking images.

[1]We are grateful to Mike Caplinger for providing these images.

They have been reprojected into an aligned coordinate system and filtered to remove biases due to any large scale albedo variations. The images also have very similar lighting conditions (they were probably taken close together in time). This image pair thus doesn't represent the best possible situation for the algorithms developed in this thesis. Nevertheless, the z-only algorithm performs well.

The estimated surface from the z-only algorithm is shown in Figure 9-2 and 9-3. The figures show that the z-only algorithm produces a reasonable estimate of the Martian surface. It should be noted that the camera geometry and light source positions were given, but the reflectance map for the surface of Mars was not. The results shown are based on the assumption of a Lambertian reflectance map. The nominal slope of the estimated surface (see surface plots) is an artifact of the camera baseline orientation with respect to the planet's surface and can be removed by shifting the camera positions along their line of sight. In fact, the estimate shown is based on cameras that are shifted slightly so that the light source positions could be represented using (p_s, q_s) gradient components.[2]

While it is difficult to see in the halftone reproduction of the estimated images, the images contain a slight ghosting effect.[3] The ghosting is like a double image and is depth dependent. By running the algorithm with slightly shifted camera principal points, different parts of the estimated images can be brought into registration. The ghosting is probably caused by using Lambertian reflectance instead of the true radiance function for the Martian surface. However, even though the algorithm used an inaccurate reflectance function, the surface estimate does contain many of the small and large scale features found in the Viking images. Examples include the cliff in the upper right and the valley just below, as well as the craters near the bottom of the images and the ridges in the center. An even better estimate of the Martian surface could have been obtained if the true reflectance function were used with the z-only algorithm.

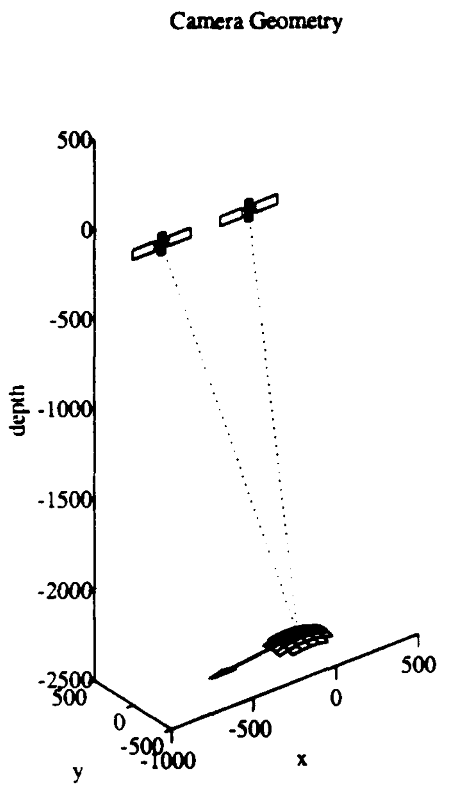
9.2 Future research

There are several directions this research can go in the future:

- Apply the fusion methodology outlined in this thesis to other vision problems.
- Apply the z-only algorithm to more real images to further judge its performance. Particularly apply the algorithm to image sets with differing lighting conditions.
- Determine the performance of the z-only algorithm when more realistic reflectance maps are used with the algorithm. Recall that while I only tested the

[2]A requirement for the particular implementation I have used. With slight modifications the algorithms could be used with any other light source position parameterization.

[3]The extent of the ghosting is less than 2 pixels for a 257-by-257 image.



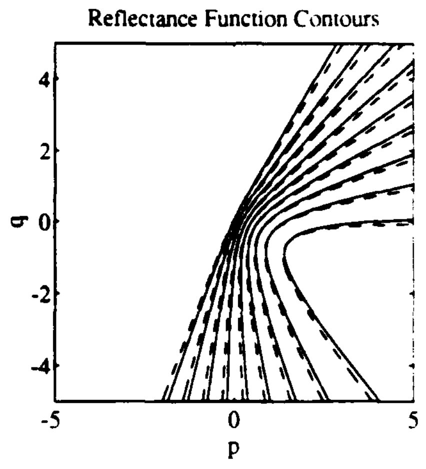


Figure 9-1: Stereo images of Mars taken by the Viking probe.

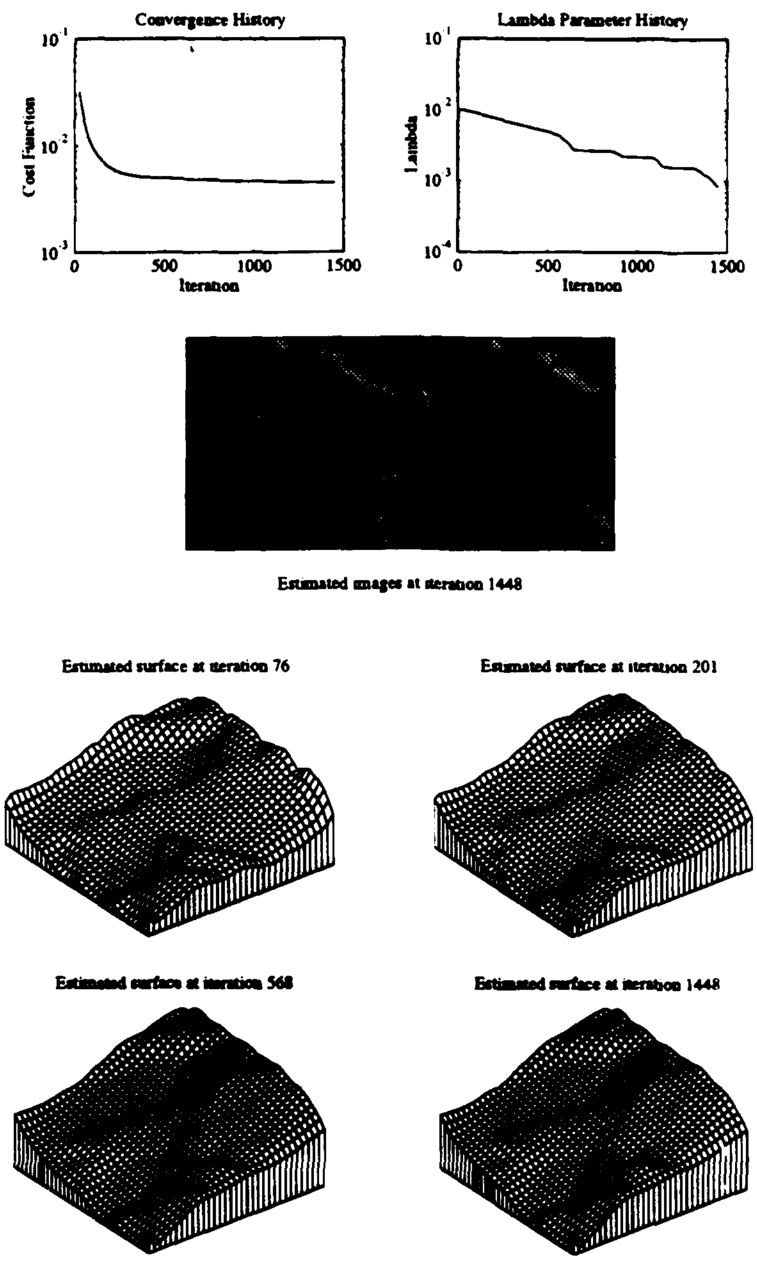


Figure 9-2: Results of running the z-only algorithm on the Viking Mars images.

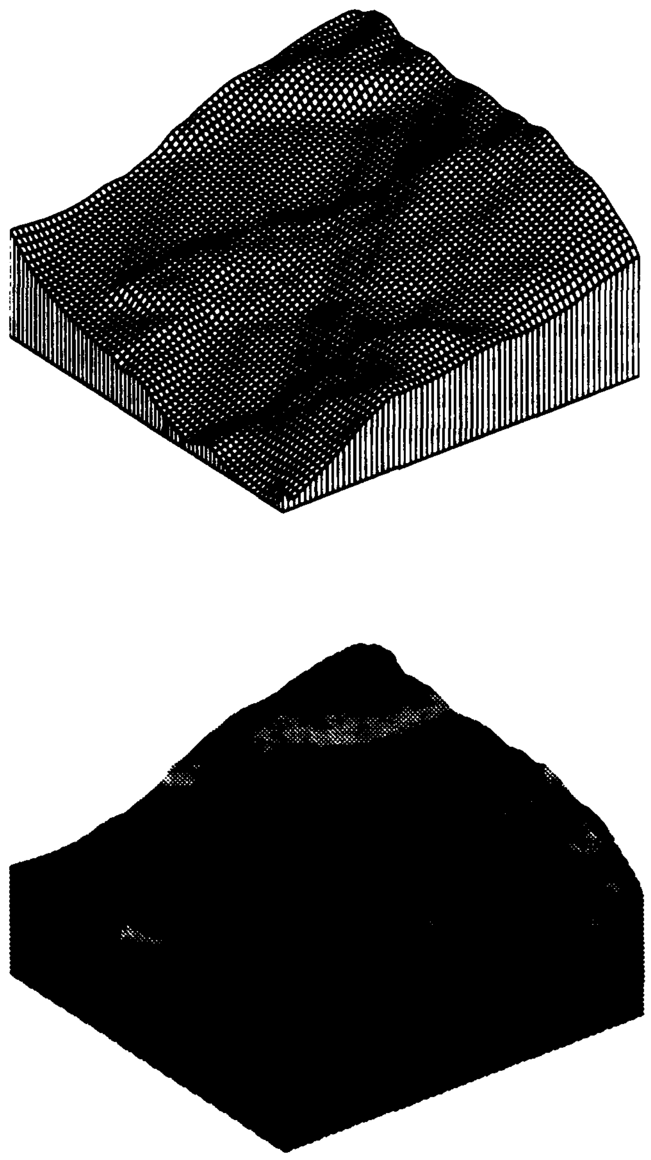


Figure 9-3: High resolution mesh and surface plot of estimated Mars surface. The surface plot was created by coloring the surface using the left estimated image.

algorithms on surfaces with Lambertian reflectance. other reflectance functions can easily be used with the algorithm.

- Develop an algorithm with better stereo integration that can correctly estimate the surface for the Hard Crater test case. To do this may require developing a new innovative stereo algorithm.

Of these possible directions, the last one presents the most challenges and offers the most rewards.

9.3 Conclusions

This thesis has shown that the variational approach to fusion problems can produce robust. well performing vision algorithms. It has also shown that fused algorithms can have significant performance gains over non-fused alternatives.

The four example algorithms, each based on a different variable representation. showed that choosing the right variable representation is important to achieving good estimates.

Finally in conjunction with this research. a new algorithm. the z-only algorithm. was developed to solve the photo-topography problem. This is the first well performing algorithm to solve that problem.

Appendix A

Gradient derivation for z-only algorithm

This appendix presents the gradient derivation for the z-only algorithm for use with the conjugate gradient optimization technique.

A.1 Cost function.

In the main text, the discrete approximation to the cost function for the z-only algorithm is given as

$$\min_{z} J = \frac{1}{2MN\epsilon^2} \sum_{x,y\in\mathcal{D}} \left\{ \left(F^{(1)}(x,y) - R^{(1)}(p,q)\right)^2 + \left(F^{(2)}(x,y) - R^{(2)}(p,q)\right)^2 + \lambda\left[z_{xx}^2 + 2z_{xy}^2 + z_{yy}^2\right]\right\} \tag{A.1}$$

where $\mathcal{D}$ is the discrete domain of the underlying variables in the global coordinate system. p and q are computed using

$$p = \frac{fz_x}{xz_x + z}, \qquad q = \frac{fz_y}{yz_y + z}. \tag{A.2}$$

M and N are the row and column dimensions of the discrete domain and ϵ is the grid spacing (assumed to the same in both the x and y directions). The $F^{(i)}(x,y)$ are interpolated from the input images $E^{(i)}(x,y)$ via linear interpolation.

$$F^{(i)}(x,y) = E^{(i)}(\bar{x},y) + (x \pm \frac{fb}{2z} - \bar{x})\left[E^{(i)}(\bar{x}+1,y) - E^{(i)}(\bar{x},y)\right] \tag{A.3}$$

where

$$\bar{x} = \text{floor}(x \pm \frac{fb}{2z}). \tag{A.4}$$

and the floor(x) function returns the greatest integer that is smaller than x. These equations assume that x is sampled on unity-spaced grid.

A.2 Bicubic extrapolation

Recall that the matched-grid implementation is used for the z-only algorithm. In this implementation, z is the same size as the image arrays $E^{(i)}$ and the boundary is extended if necessary based on bicubic interpolation. Let z_{i1} be a value in the first column of z (i.e., on the left boundary of z). Then the extrapolated value z_{i0}, is computed using

$$z_{i0} = 3z_{i1} - 3z_{i2} + z_{i3}. \tag{A.5}$$

Similar equations work for the other boundaries of z. In this way, a M-by-N matrix can be extrapolated to form a $(M+2)$-by-$(N+2)$ matrix.

A.3 Gradient of terms based on convolution.

Theorem 1 *Let y be computed via a 2-D convolution from the field z and the filter h, $y = z * h$. Let $f(x) : \Re \rightarrow \Re$, be any scalar point function. Then the derivative of the scalar cost function $J = \sum_i \sum_j f(y_{ij})$ with respect to z_{lm} is*

$$\frac{\partial J}{\partial z_{lm}} = \left.\frac{df(x)}{dx}\right|_{x=y_{ij}} * \tilde{h} \tag{A.6}$$

where

$$\tilde{h}_{ij} = h_{-i,-j}. \tag{A.7}$$

Proof Using the definition of the 2-D convolution,

$$y_{ij} = \sum_k \sum_s z_{ks} h_{i-k,j-s}, \tag{A.8}$$

the cost function can be expanded to obtain,

$$J = \sum_i \sum_j f\left(\sum_k \sum_s z_{ks} h_{i-k,j-s}\right). \tag{A.9}$$

The sums are taken over the entire range of indices into the matrices. Taking the derivative and using the chain rule we find.

$$\frac{\partial J}{\partial z_{lm}} = \sum_i \sum_j \frac{df}{dx} \frac{\partial}{\partial z_{lm}} \left[\sum_k \sum_s z_{ks} h_{i-k,j-s} \right] \tag{A.10}$$

$$= \sum_i \sum_j \frac{df}{dx} \sum_k \sum_s \frac{\partial z_{ks}}{\partial z_{lm}} h_{i-k,j-s} \tag{A.11}$$

$$= \sum_i \sum_j \frac{df}{dx} h_{i-l,j-m} \tag{A.12}$$

$$= \sum_i \sum_j \frac{df}{dx} \bar{h}_{l-i,m-j} \tag{A.13}$$

$$= \frac{df}{dx} * \bar{h} \quad \blacksquare \tag{A.14}$$

The result holds for computational molecules as well since operating with a computational molecule m is equivalent to convolving with $\tilde{m}$.

Theorem 2 *Theorem 1 also holds for scalar functions of multiple arguments. In that case.*

$$\frac{\partial J}{\partial z} = \frac{\partial f}{\partial x_1} * \bar{h} + \frac{\partial f}{\partial x_2} * \bar{g}. \tag{A.15}$$

Proof Suppose y and w are defined via 2-D convolution. $y = z * h$, $w = z * g$, and $f(x_1, x_2) : [\Re \times \Re] \rightarrow \Re$, then the partial derivative of the cost function $J = \sum_i \sum_j f(y_{ij}, w_{ij})$ is

$$\frac{\partial J}{\partial z} = \frac{\partial f}{\partial x_1} * \bar{h} + \frac{\partial f}{\partial x_2} * \bar{g}. \tag{A.16}$$

The proof follows the same lines as the proof for Theorem 1 ■.

A.4 Cost function derivative.

Using the chain rule, the derivative of the cost function (Equation A.1) is

$$\begin{aligned} \frac{dJ}{dz_{km}} = \sum_x \sum_y \Bigg\{ & (F^{(1)} - R^{(1)}) \left(\frac{\partial F^{(1)}}{\partial z_{km}} - R_p^{(1)} \frac{\partial p}{\partial z_{km}} - R_q^{(1)} \frac{\partial q}{\partial z_{km}} \right) \\ & + (F^{(2)} - R^{(2)}) \left(\frac{\partial F^{(1)}}{\partial z_{km}} - R_p^{(2)} \frac{\partial p}{\partial z_{km}} - R_q^{(2)} \frac{\partial q}{\partial z_{km}} \right) \\ & + \lambda \left[z_{xx} \frac{\partial z_{xx}}{\partial z_{km}} + 2 z_{xy} \frac{\partial z_{xy}}{\partial z_{km}} + z_{yy} \frac{\partial z_{yy}}{\partial z_{km}} \right] \Bigg\}. \end{aligned} \tag{A.17}$$

Let's look at each term in turn. The derivatives involving the interpolated images can be computed using the definition of $F^{(i)}$.

$$\frac{\partial F}{\partial z_{km}} = E_x(\bar{x},y)\frac{\partial \bar{x}}{\partial z_{km}} + \left(\mp\frac{fb}{2z^2} - \frac{\partial \bar{x}}{\partial z_{km}}\right)[E(\bar{x}+1,y) - E(\bar{x},y)]$$
$$+ \left(x \pm \frac{fb}{2z^2} - \bar{x}\right)[E_x(\bar{x}+1,y) - E_x(\bar{x},y)]\frac{\partial \bar{x}}{\partial z_{km}} \quad (A.18)$$
$$= \mp\frac{fb}{2z^2}E_x(\bar{(x)},y). \quad (A.19)$$

That last equality is possible since $\partial\bar{x}/\partial z_{km} = 0$ for one-sided derivatives.

The derivatives of the gradient components p and q can also be computed using their definition. They are,

$$\frac{\partial p}{\partial z_{km}} = \frac{f\frac{\partial z_x}{\partial z_{km}}(xz_x + z) - fz_x(x\frac{\partial z_x}{\partial z_{km}} + \frac{\partial z}{\partial z_{km}})}{(xz_x + z)^2}, \quad (A.20)$$
$$= \frac{fz\frac{\partial z_x}{\partial z_{km}} - fz_x\frac{\partial z}{\partial z_{km}}}{(xz_x + z)^2}, \quad (A.21)$$

and,

$$\frac{\partial q}{\partial z_{km}} = \frac{fz\frac{\partial z_y}{\partial z_{km}} - fz_y\frac{\partial z}{\partial z_{km}}}{(yz_y + z)^2}. \quad (A.22)$$

As discussed in the main text, the partial derivatives of z (z_x, z_y, z_{xx}, and so on), are computed via 2-D computational molecules. Call these molecules, h_x, h_y, h_{xx}, and so on, in the obvious way. Using the results of Theorem 1, the derivative of the cost function can then be written,

$$\frac{\partial J}{\partial z_{km}} = \left[-(F^{(1)} - R^{(1)})E_x^{(1)} + (F^{(2)} - R^{(2)})E_x^{(2)}\right]\frac{fb}{2z^2}$$
$$+ \left[(F^{(1)} - R^{(1)})R_p^{(1)} + (F^{(2)} - R^{(2)})R_p^{(2)}\right]\frac{fz_x}{(xz_x + z)^2}$$
$$+ \left[(F^{(1)} - R^{(1)})R_q^{(1)} + (F^{(2)} - R^{(2)})R_q^{(2)}\right]\frac{fz_y}{(yz_y + z)^2}$$
$$- \left\{\left[(F^{(1)} - R^{(1)})R_p^{(1)} + (F^{(2)} - R^{(2)})R_p^{(2)}\right]\frac{fz}{(xz_x + z)^2}\right\} \diamond \bar{h}_x$$
$$- \left\{\left[(F^{(1)} - R^{(1)})R_q^{(1)} + (F^{(2)} - R^{(2)})R_q^{(2)}\right]\frac{fz}{(yz_y + z)^2}\right\} \diamond \bar{h}_y$$
$$+ \lambda\left[z_{xx} \diamond \bar{h}_{xx} + 2z_{xy} \diamond \bar{h}_{xy} + z_{yy} \diamond \bar{h}_{yy}\right]. \quad (A.23)$$

These equations are not the whole story since they don't work on the boundary. I will show how to take into account the effect of the bicubic extrapolation on the left boundary: the other boundaries are dealt with in a similar way. Suppose h has either 2 or 3 columns, and let $\tilde{z}$ be the extrapolated version of z (for simplicity assume z is square of size N-by-N). In this case the subscripts on z run from 0 to $N-1$. Theorem 1 states that the derivative of $J = \sum\sum f(\tilde{z} * h)$ is

$$\frac{\partial J}{\partial \tilde{z}} = \frac{df}{dx}\tilde{z} * \bar{h}. \tag{A.24}$$

Call this matrix $\tilde{G}$. Since $\tilde{z}$ is computed from z via the bicubic extrapolation described in Section A.2, we can compute $\partial J/\partial z$ by taking into account the dependence of z_{i0} on z_{i1}, z_{i2}, and z_{i3}. Let G be the matrix $\partial J/\partial z$, then columns 1–3 of G can be computed using

$$G_{i1} = \tilde{G}_{i1} + 3\tilde{G}_{i0}, \tag{A.25}$$
$$G_{i2} = \tilde{G}_{i2} - 3\tilde{G}_{i0}, \tag{A.26}$$
$$G_{i3} = \tilde{G}_{i3} + \tilde{G}_{i0}. \tag{A.27}$$

Similar expressions hold for the other boundaries.

Appendix B

M-file Listings

This appendix contains the MATLAB[1] M-file source files for the DFSS algorithms. The source files are presented for the six algorithms in this thesis: zpq, z-only, dual z, disparity, sfs, and varying albedo. A key to the files is given in Table B-1.

B.1 Cost function and gradient routines

B.1.1 hbdfss_cost3c.m

```
function [J,FR1,FR2]=hbdfss_cost(v,zsize,levels,params,E1,E2,lambda,mu)
%HBDFSS_COST3C Cost function for depth from shading and stereo
%   problem using hierarchical basis functions. Uses true
%   perspective projection.
%
%   J=HBDFSS_COST3C(V,ZSIZE,LEVELS,PARAMS,E1,E2,LAMBDA,MU) where
%   V = [z,p,q] are the optimization variables, ZSIZE is
%   SIZE(z), LEVELS is the number of h-basis levels. E1 and E2 are
%   the input images, LAMBDA is the scalar weighting factor on
%   departure from smoothness and MU is the scalar weighting factor on
%   integrability. The image parameters
%   PARAMS = [f,b,z0,ps1,qs1,ps2,ps2,vx1,vy1,vx2,vy2].

%   Clay M. Thompson 5-18-92
% Revised to use correct p,q calculation.
% Revised to use p,q,z of the same size as E.
% Revised to output estimated image.
% Revised to support multi-grid scheme.

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2;
delta = params(12);
area = prod(zsize)*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
```

[1]MATLAB is a product of The MathWorks, Inc., 24 Prime Park Way, Cochituate Place, Natick MA 01760, (508)653-1415.

Algorithm cost functions and gradient routines

hbdfss_cost3c.m	Cost function for zpq algorithm.
hbdfss_grad3c.m	Gradient function for zpq algorithm.
hbdfss_cost2c.m	Cost function for z-only algorithm.
hbdfss_grad2c.m	Gradient function for z-only algorithm.
hbdfss_cost4.m	Cost function for dual-z algorithm.
hbdfss_grad4.m	Gradient function for dual-z algorithm.
hbdfss_cost7.m	Cost function for disparity algorithm.
hbdfss_grad7.m	Gradient function for disparity algorithm.
hbdfss_cost8b.m	Cost function for varying albedo algorithm.
hbdfss_grad8b.m	Gradient function for varying albedo algorithm.
hbsfs_cost.m	Cost function for shape-from-shading algorithm.
hbsfs_grad.m	Gradient function for shape-from-shading algorithm.

Support routines

rmap	Lambertian reflectance function.
rmapp	Derivative of reflectance function with respect to p.
rmapq	Derivative of reflectance function with respect to q.
conjgrad.m	*Conjugate gradient optimization.*
lsearch.m	Line search function for conjugate gradient optimization.
filter2d.m	2-D computational molecule filtering.
cfilter2d.m	2-D computational molecule filtering with bicubic interpolation.
hbasis.m	Main level hierarchical basis conversion.
hb.m	Hierarchical basis interpolation.
hbt.m	Adjoint hierarchical basis interpolation.
interpx.m	Linear interpolation in the x direction.
domain2d.m	2-D plaid domain generation.
icubic.m	1-D cubic interpolation.
dcubicx.m	Derivative of 1-D cubic interpolation with respect to x.
dcubicz.m	Derivative of 1-D cubic interpolation with respect to z.

Table B-1: M-file Descriptions.

```
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Extract z,p,q and transform into nodal basis.
mz = zsize(1); nz = zsize(2);
z = v(:,1:nz);
p = v(:,nz+[1:nz]);
q = v(:,2*nz + [1:nz]);
z(:) = hbasis(z,levels);
p(:) = hbasis(p,levels);
q(:) = hbasis(q,levels);

% Spacial coordinates in image.
[mz,nz] = size(z);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hzz = [-1 0 1]/(2*delta); % X-derivative for zx
hyz = [1;0;-1]/(2*delta); % Y-derivative for zy
hx = [-1 1;-1 1]/(2*delta);     % X-derivative
hy = [1 1;-1 -1]/(2*delta); % Y-derivative

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);

% Compute numerical derivatives of p,q, and z.
px = filter2d(p,hx,'resize');
py = filter2d(p,hy,'resize');
qx = filter2d(q,hx,'resize');
qy = filter2d(q,hy, resize');
zx = cfilter2d(z,hzz);
zy = cfilter2d(z,hyz);

% Compute estimates of p,q using numerical derivatives of z.
pe = f*zx./(x.*zx+z);
qe = f*zy./(y.*zy+z);

% Compute disparity.
d = (f*b/2)./z;

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,(x+d-vx1-1)/delta+1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end,
F2 = interpx(E2,(x-d-vx2-1)/delta+1);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end,

term1 = (0.5/area)*sum(sum((F1-R1).^2 + (F2-R2).^2));
term2 = (0.5/area)*lambda*sum(sum(px.^2 + py.^2 + qx.^2 + qy.^2));
term3 = (0.5/area)*mu*sum(sum((pe-p).^2 + (qe-q).^2)),
%disp(sprintf('Terms: %12.5f %12.5f %12.5f',term1,term2,term3));
J = [term1+term2+term3,term1,term2,term3];

if nargout>1,
  FR1 = [E1;F1;R1;abs(F1-R1)];
  FR2 = [E2;F2;R2;abs(F2-R2)];
end

```

B.1.2 hbdfss_grad3c.m

```
function [J,FR1,FR2]=hbdfss_cost(v,zsize,levels,params,E1,E2,lambda,mu)
%HBDFSS_COST3C Cost function for depth from shading and stereo
%   problem using hierarchical basis functions. Uses true
%   perspective projection.
%
%   J=HBDFSS_COST3C(V,ZSIZE,LEVELS,PARAMS,E1,E2,LAMBDA,MU) where
%   V = [z,p,q] are the optimization variables, ZSIZE is
%   SIZE(z), LEVELS is the number of h-basis levels. E1 and E2 are
%   the input images, LAMBDA is the scalar weighting factor on
%   departure from smoothness and MU is the scalar weighting factor on
%   integrability. The image parameters
%   PARAMS = [f,b,z0,ps1,qs1,ps2,qs2,vx1,vy1,vx2,vy2].

%   Clay M. Thompson 5-18-92
% Revised to use correct p,q calculation.
% Revised to use p,q,z of the same size as E.
% Revised to output estimated image.
% Revised to support multi-grid scheme.

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2;
delta = params(12);
area = prod(zsize)*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Extract z,p,q and transform into nodal basis.
mz = zsize(1); nz = zsize(2);
z = v(:,1:nz);
p = v(:,nz+[1:nz]);
q = v(:,2*nz + [1:nz]);
z(:) = hbasis(z,levels);
p(:) = hbasis(p,levels);
q(:) = hbasis(q,levels);

% Spacial coordinates in image.
[mz,nz] = size(z);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hxz = [-1 0 1]/(2*delta); % X-derivative for zx
hyz = [1;0;-1]/(2*delta); % Y-derivative for zy
hx = [-1 1;-1 1]/(2*delta);     % X-derivative
hy = [1 1;-1 -1]/(2*delta); % Y-derivative

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);

% Compute numerical derivatives of p,q, and z.
px = filter2d(p,hx,'resize');
py = filter2d(p,hy,'resize');
qx = filter2d(q,hx,'resize');
qy = filter2d(q,hy,'resize');
zx = cfilter2d(z,hxz);
zy = cfilter2d(z,hyz);

% Compute estimates of p,q using numerical derivatives of z.
pe = f*zx./(x.*zx+z);
qe = f*zy./(y.*zy+z);

% Compute disparity.
d = (f*b/2)./z;

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,(x+d-vx1-1)/delta+1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end,
F2 = interpx(E2,(x-d-vx2-1)/delta+1);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end,

term1 = (0.5/area)*sum(sum((F1-R1).^2 + (F2-R2).^2));
term2 = (0.5/area)*lambda*sum(sum(px.^2 + py.^2 + qx.^2 + qy.^2));
term3 = (0.5/area)*mu*sum(sum((pe-p).^2 + (qe-q).^2));
%disp(sprintf('Terms: %12.5f %12.5f %12.5f',term1,term2,term3));
J = [term1+term2+term3,term1,term2,term3];

if nargout>1,
  FR1 = [E1;F1;R1;abs(F1-R1)];
  FR2 = [E2;F2;R2;abs(F2-R2)];
end

```

B.1.3 hbdfss_cost2c.m

```
function [J,FR1,FR2]=hbdfss_cost(z,n,params,E1,E2,lambda)
%HBDFSS_COST2C Cost function for depth from shading and stereo
%   problem with heirarchical basis functions. Uses true
%   perspective projection.
```

```
%
%    J=HBDFSS_COST2C(Z,N,PARAMS,E1,E2,LAMBDA) where Z is the surface
%    height, N is the number of H-basis levels  E1 and E2 are the input
%    images, LAMBDA is the scalar weighting factor on departure from
%    smoothness. The image parameters PARAMS = [f,b,z0,ps1,qs1,ps2,ps2].
%
%    J= [Jtot,Jimage,Jsmooth].

%    Clay M. Thompson 2-24-91
%    Revised 4-3-91 by cmt
%    Revised 5-10-91 by cmt
%    Revised 5-19-91 by cmt
%    Revised 4-30-92 by cmt to support multi-grid scheme.
% Revised to output estimated image.

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2/z0;
delta = params(12);
area = prod(size(z))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(z);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/(2*delta);     % X-derivative
hy = [1;0;-1]/(2*delta);     % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/(4*delta*delta);
%hxy = [-1 0 1;0 0 0;1 0 -1]/(4*delta*delta);
hxy = [-1 1;1 -1]/(delta*delta);
hyy = [1 2 1;-2 -4 -2;1 2 1]/(4*delta*delta);
del2 = [1 4 1;4 -20 4;1 4 1]/(6*delta*delta);

% Compute p,q using numerical derivatives of z.
z = hbasis(z,n);
zx = cfilter2d(z,hx);
zy = cfilter2d(z,hy);
p = f*zx ./(x.*zx+z);
q = f*zy ./(y.*zy+z);

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);

% Compute numerical derivatives of z.
zxx = cfilter2d(z,hxx);
zxy = cfilter2d(z,hxy);
zyy = cfilter2d(z,hyy);

% Compute disparity.
d = (f*b/2)./z;

% Determine stereo mapped images F1,F2  Set error to zero where F is NaN
F1 = interpx(E1,(x+d-vx1-1)/delta+1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end, keep1 = out;
F2 = interpx(E2,(x-d-vx2-1)/delta+1);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end, keep2 = out;

term1 = (0.5/area)*sum(sum((F1-R1).^2 + (F2-R2).^2));
term2 = (0.5/area)*lambda*(sum(sum(zxx.^2)) + 2*sum(sum(zxy.^2)) + sum(sum(zyy.^2)));
%term2 = (0.5/area)*lambda*sum(sum(delsqz.^2));
%disp(sprintf('Terms: %12.5f %12.5f',term1,term2));
J = [term1+term2,term1,term2];

if nargout>1,
  FR1 = [E1;F1;R1;abs(F1-R1);keep1];
  FR2 = [E2;F2;R2;abs(F2-R2);keep2];
```

```
end
```

B.1.4 hbdfss_grad2c.m

```
function G=hbdfss_grad(z,n,params,E1,E2,lambda)
%HBDFSS_GRAD2C Gradient function for depth from shading and stereo
%   problem with heirarchical basis functions. Uses true perspective
%   projection.
%
%   G=HBDFSS_GRAD2C(Z,N,PARAMS,E1,E2,LAMBDA) where Z is the height
%   map, N is the number of H-basis levels. E1 and E2 are the input
%   images, LAMBDA is the scalar weighting factor on departure from
%   smoothness. The image parameters PARAMS = [f,b,z0,ps1,qs1,ps2,ps2];

%   Clay M. Thompson 2-24-91
%   Revised 4-3-91 by cmt.
%   Revised 5-10-91 by cmt.
%   Revised 5-19-91 by cmt.
%   Revised 4-30-92 by cmt for multi-grid scheme.

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2/z0;
delta = params(12);
area = prod(size(z))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(z);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/2/delta;  % X-derivative
hy = [1;0 -1]/2/delta;  % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/4/delta/delta;
%hxy = [-1 0 1;0 0 0;1 0 -1]/4/delta/delta;
hxy = [-1 1;1 -1]/delta/delta;
hyy = [1 2 1;-2 -4 -2;1 2 1]/4/delta/delta;
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta;

% Compute p,q using numerical derivatives of z.
z = hbasis(z,n);
zx = cfilter2d(z,hx);
zy = cfilter2d(z,hy);
p = f*zx./(x.*zx+z);
q = f*zy./(y.*zy+z);

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);
Rp1 = rmapp(p,q,ps1,qs1);
Rp2 = rmapp(p,q,ps2,qs2);
Rq1 = rmapq(p,q,ps1,qs1);
Rq2 = rmapq(p,q,ps2,qs2);

% Compute numerical derivatives of z.
zxx = cfilter2d(z,hxx);
zxy = cfilter2d(z,hxy);
zyy = cfilter2d(z,hyy);

% Compute disparity.
d = (f*b/2)./z;
x1 = (x+d-vx1-1)/delta+1;
x2 = (x-d-vx2-1)/delta+1;

% Compute 1st difference in the x direction of image arrays.
% Based on the stencil:   -1 1
[m1,n1] = size(E1); [m2,n2] = size(E2);
```

```
stencil = [-1 1]/delta;
Ex1 = filter2d(E1,stencil,'resize');
Ex2 = filter2d(E2,stencil,'resize');

% Determine stereo mapped image derivatives.
Fx1 = interpx(Ex1,floor(x1)); zz = zeros(m1,n1);
out = isnan(Fx1); if any(out(:)), Fx1(out) = zz(out); end
Fx2 = interpx(Ex2,floor(x2)); zz = zeros(m2,n2);
out = isnan(Fx2); if any(out(:)), Fx2(out) = zz(out); end

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,x1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end
F2 = interpx(E2,x2);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end

% Compute Gradient
G = zeros(mz,nz);
ERp1 = (F1-R1).*Rp1; ERq1 = (F1-R1).*Rq1;
ERp2 = (F2-R2).*Rp2; ERq2 = (F2-R2).*Rq2;
denx = (x.*zx+z).^2;
deny = (y.*zy+z).^2;
G = -(f*b/2)*((F1-R1).*Fx1 - (F2-R2).*Fx2)./(z.^2) + ...
    f*((ERp1+ERp2).*zx./denx + (ERq1+ERq2).*zy./deny);
G(:) = G - cfilter2d((ERp1+ERp2).*z./denx,-f*hx,'grad');
G(:) = G - cfilter2d((ERq1+ERq2).*z./deny,-f*hy,'grad');
G(:) = G + cfilter2d(zxx,lambda*hxx,'grad') + ...
    cfilter2d(zxy,2*lambda*hxy,'grad') + ...
    cfilter2d(zyy,lambda*hyy,'grad');
G = hbasis(G*(1/area),n,'trans');
```

B.1.5 hbdfss_cost4.m

```
function [J,FR1,FR2]=hbdfss_cost4(v,levels,params,E1,E2,lambda,mu)
%HBDFSS_COST4 Cost function for depth from shading and stereo
%   problem using hierarchical basis functions. Uses true
%   perspective projection and dual z maps.
%
%   J=HBDFSS_COST4(V,LEVELS,PARAMS,E1,E2,LAMBDA,MU) where
%   V = [z1(:);z2(:)] are the optimization variables, LEVELS is the
%   number of h-basis levels. E1 and E2 are the input images, LAMBDA
%   is the scalar weighting factor on departure from smoothness and MU
%   is the scalar weighting factor on stereo matching. The image
%   parameters are PARAMS = [f,b,z0,ps1,qs1,ps2,ps2].
%
% J = [Jtot,Jimage,Jsmooth,Jstereo]

%   Clay M. Thompson 7-18-91
% Revised to support multi-grid scheme

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
delta = params(12);
area = prod(size(E1))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);

% Extract z1 & z2 and transform into nodal basis.
[mz1,nz1] = size(E1); mz1 = mz1+2; nz1 = nz1+2;
[mz2,nz2] = size(E2); mz2 = mz2+2; nz2 = nz2+2;
z1 = zeros(mz1,nz1); z1(:) = v(1:mz1*nz1);
z2 = zeros(mz2,nz2); z2(:) = v(mz1*nz1+[1:mz2*nz2]);
z1(:) = hbasis(z1,levels);
z2(:) = hbasis(z2,levels);

% Spacial coordinates in image.
[x1,y1] = domain2d([0:mz1-1]*delta+vx1+1,[0:nz1-1]*delta+vy1+1);
[x2,y2] = domain2d([0:mz1-1]*delta+vx2+1,[0:nz1-1]*delta+vy2+1);

% Stencils
hx = [-1 0 1]/2/delta;  % X-derivative
```

```
hy = [1;0;-1]/2/delta;  % Y-derivative
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta;

% Compute p,q using numerical derivatives of z
zx1 = filter2d(z1,hx,'resize'); zy1 = filter2d(z1,hy,'resize');
zx2 = filter2d(z2,hx,'resize'); zy2 = filter2d(z2,hy,'resize');
rows = 2:mz1-1; cols = 2:nz1-1;
p1 = f*zx1(rows,:)./(x1(rows,cols).*zx1(rows,:)+z1(rows,cols));
q1 = f*zy1(:,cols)./(y1(rows,cols).*zy1(:,cols)+z1(rows,cols));
rows = 2:mz2-1; cols = 2:nz2-1;
p2 = f*zx2(rows,:)./(x2(rows,cols).*zx2(rows,:)+z2(rows,cols));
q2 = f*zy2(:,cols)./(y2(rows,cols).*zy2(:,cols)+z2(rows,cols));

% Compute reflectance map values.
R1  = rmap(p1,q1,ps1,qs1);
R2  = rmap(p2,q2,ps2,qs2);

% Compute interpolated images.  Set error to zero where zbar is NaN.
zbar1 = icubic(x1(1,:)',z1',(x2+(f*b)./z2)')';
out = isnan(zbar1); if any(out(:)), zbar1(out) = z2(out); end
zbar2 = icubic(x2(1,:)',z2',(x1-(f*b)./z1)')';
out = isnan(zbar2); if any(out(:)), zbar2(out) = z1(out); end

term1 = (0.5/area)*sum(sum((E1-R1).^2 + (E2-R2).^2));   % sfs
term2 = (0.5/area)*lambda*sum(sum( ...
    filter2d(z1,del2,'resize').^2 + filter2d(z2,del2,'resize').^2  )); % Smooth
term3 = (0.5/area)*mu*sum(sum((z1-zbar2).^2 + (zbar1-z2).^2)); % Stereo
%disp(sprintf('Terms: %12.5f %12.5f %12.5f',term1,term2,term3));
J = [term1+term2+term3,term1,term2,term3];

if nargout>1,
  FR1 = [E1;R1;abs(E1-R1)];
  FR2 = [E2;R2;abs(E2-R2)];
end
```

B.1.6 hbdfss_grad4.m

```
function G=hbdfss_grad4(v,levels,params,E1,E2,lambda,mu)
%HBDFSS_GRAD4 Gradient function for depth from shading and stereo
%   problem using hierarchical basis representation. Uses true
%   perspective projection and dual z maps.
%
%   G=HBDFSS_GRAD4(V,LEVELS,PARAMS,E1,E2,LAMBDA,MU) where
%   V = [z1(:);z2(:)] are the optimization variables, LEVELS is the
%   number of h-basis levels. E1 and E2 are the input images, LAMBDA
%   is the scalar weighting factor on departure from smoothness and MU
%   is the scalar weighting factor on stereo matching. The image
%   parameters are PARAMS = [f,b,z0,ps1,qs1,ps2,ps2].

%   Clay M. Thompson 7-18-91

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
delta = params(12);
area = prod(size(E1))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);

% Extract z1 & z2 and transform into nodal basis.
[mz1,nz1] = size(E1); mz1 = mz1+2; nz1 = nz1+2;
[mz2,nz2] = size(E2); mz2 = mz2+2; nz2 = nz2+2;
z1 = zeros(mz1,nz1); z1(:) = v(1:mz1*nz1);
z2 = zeros(mz2,nz2); z2(:) = v(mz1*nz1+[1:mz2*nz2]);
z1(:) = hbasis(z1,levels);
z2(:) = hbasis(z2,levels);

% Spacial coordinates in image.
[x1,y1] = domain2d([0:mz1-1]*delta+vx1+1,[0:nz1-1]*delta+vy1+1);
[x2,y2] = domain2d([0:mz1-1]*delta+vx2+1,[0:nz1-1]*delta+vy2+1);

% Stencils
hx = [-1 0 1]/2/delta;  % X-derivative
```

```
hy = [1;0;-1]/2/delta;  % Y-derivative
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta; % Laplacian

% Compute p,q using numerical derivatives of z
zx1 = filter2d(z1,hx,'resize'); zy1 = filter2d(z1,hy,'resize');
zx2 = filter2d(z2,hx,'resize'); zy2 = filter2d(z2,hy,'resize');
rows = 2:mz1-1; cols = 2:nz1-1;
p1 = f*zx1(rows,:)./(x1(rows,cols).*zx1(rows,:)+z1(rows,cols));
q1 = f*zy1(:,cols)./(y1(rows,cols).*zy1(:,cols)+z1(rows,cols));
rows = 2:mz2-1; cols = 2:nz2-1;
p2 = f*zx2(rows,:)./(x2(rows,cols).*zx2(rows,:)+z2(rows,cols));
q2 = f*zy2(:,cols)./(y2(rows,cols).*zy2(:,cols)+z2(rows,cols));

% Compute reflectance map values.
R1  = rmap(p1,q1,ps1,qs1);
Rp1 = rmapp(p1,q1,ps1,qs1); Rq1 = rmapq(p1,q1,ps1,qs1);
R2  = rmap(p2,q2,ps2,qs2);
Rp2 = rmapp(p2,q2,ps2,qs2); Rq2 = rmapq(p2,q2,ps2,qs2);

% Compute interpolated images and related terms.  Set gradient to zero where zbar is NaN.
d = (f*b)./z1;
zbar2 = icubic(x2(1,:)',z2',(x1-d)')';
out = find(isnan(zbar2));
zbarx2 = dcubicx(x2(1,:)',z2',(x1-d)')'/delta;
if length(out)>0, zbar2(out) = z1(out); zbarx2(out) = 0*z1(out); end
zbarz2 = dcubicz(x2(1,:)',z2',(x1-d)',(z1-zbar2)')';

d = (f*b)./z2;
zbar1 = icubic(x1(1,:)',z1',(x2+d)')';
out = find(isnan(zbar1));
zbarx1 = dcubicx(x1(1,:)',z1',(x2+d)')'/delta;
if length(out)>0, zbar1(out) = z2(out); zbarx1(out) = 0*z2(out); end
zbarz1 = dcubicz(x1(1,:)',z1',(x2+d)',(zbar1-z2)')';

% Compute error terms

% Now form gradient
rows = 2:mz1-1; cols = 2:nz1-1;
ERp = (E1-R1).*Rp1; ERq = (E1-R1).*Rq1;
denx = (x1(rows,cols).*zx1(rows,:)+z1(rows,cols)).^2;
deny = (y1(rows,cols).*zy1(:,cols)+z1(rows,cols)).^2;
dJdz1 = zeros(mz1,nz1);
dJdz1(rows,:) = -filter2d(ERp.*z1(rows,cols)./denx,-f*hx);
dJdz1(:,cols) = dJdz1(:,cols) - ...
    filter2d(ERq.*z1(rows,cols)./deny,-f*hy);
dJdz1(rows,cols) = dJdz1(rows,cols) + ...
    f*(ERp.*zx1(rows,:)./denx + ERq.*zy1(:,cols)./deny);
dJdz1(:) = dJdz1 + ...
    lambda*filter2d(filter2d(z1,del2,'resize'),del2) + ...
    mu*(z1-zbar2).*(1-(f*b)*zbarx2./(z1.^2)) + ...
    mu*zbarz1;

rows = 2:mz2-1; cols = 2:nz2-1;
ERp = (E2-R2).*Rp2; ERq = (E2-R2).*Rq2;
denx = (x2(rows,cols).*zx2(rows,:)+z2(rows,cols)).^2;
deny = (y2(rows,cols).*zy2(:,cols)+z2(rows,cols)).^2;
dJdz2 = zeros(mz2,nz2);
dJdz2(rows,:) = -filter2d(ERp.*z2(rows,cols)./denx,-f*hx);
dJdz2(:,cols) = dJdz2(:,cols) - ...
    filter2d(ERq.*z2(rows,cols)./deny,-f*hy);
dJdz2(rows,cols) = dJdz2(rows,cols) + ...
    f*(ERp.*zx2(rows,:)./denx + ERq.*zy2(:,cols)./deny);
dJdz2(:) = dJdz2 + ...
    lambda*filter2d(filter2d(z2,del2,'resize'),del2) - ...
    mu*zbarz2 - ...
    mu*(zbar1-z2).*((f*b)*zbarx1./(z2.^2)+1);

% Return gradient in h-basis coordinates
dJdz1 = hbasis(dJdz1*(1/area),levels,'trans');
dJdz2 = hbasis(dJdz2*(1/area),levels,'trans');
G = [dJdz1(:);dJdz2(:)];

```

B.1.7 hbdfss_cost7.m

```
function [J,FR1,FR2]=hbdfss_cost(u,n,params,E1,E2,lambda)
%HBDFSS_COST7 Cost function for depth from shading and stereo
%   problem with heirarchical basis functions. Uses true
%   perspective projection.
%
%   J=HBDFSS_COST7(U,N,PARAMS,E1,E2,LAMBDA) where U is the disparity,
%   and N is the number of H-basis levels.  E1 and E2 are the input
%   images, LAMBDA is the scalar weighting factor on departure from
%   smoothness. The image parameters PARAMS = [f,b,z0,ps1,qs1,ps2,ps2].
%
%   J= [Jtot,Jimage,Jsmooth].

%   Clay M. Thompson 5-23-92
%   Uses u = fb/2z - fb/2z_0 to enhance the numerical stability of the algorithm.
%   Revised to support multi-grid scheme

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
u0 = f*b/2/z0;
delta = params(12);
area = prod(size(u))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(u);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/(2*delta);     % X-derivative
hy = [1;0;-1]/(2*delta);     % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/(4*delta*delta),
%hxy = [-1 0 1;0 0 0;1 0 -1]/(4*delta*delta);
hxy = [-1 1;1 -1]/(delta*delta);
hyy = [1 2 1;-2 -4 -2;1 2 1]/(4*delta*delta);
del2 = [1 4 1;4 -20 4;1 4 1]/(6*delta*delta);

% Compute p,q using numerical derivatives of u.
u = hbasis(u,n);
ux = cfilter2d(u,hx);
uy = cfilter2d(u,hy);
p = f*ux./(x.*ux-u-u0);
q = f*uy./(y.*uy-u-u0);

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);

% Compute numerical derivatives of u.
uxx = cfilter2d(u,hxx);
uxy = cfilter2d(u,hxy);
uyy = cfilter2d(u,hyy);

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,(x+u+u0-vx1-1)/delta+1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end,
F2 = interpx(E2,(x-u-u0-vx2-1)/delta+1);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end,

%image([F1,F2;R1,R2;abs(F1-R1),abs(F2-R2)],[0 1])

term1 = (0.5/area)*sum(sum((F1-R1).^2 + (F2-R2).^2));
term2 = (0.5/area)*lambda*(sum(sum(uxx.^2)) + 2*sum(sum(uxy.^2)) + sum(sum(uyy.^2)));
%term2 = (0.5/area)*lambda*sum(sum(delsqz.^2));
%disp(sprintf('Terms: %12.5f %12.5f',term1,term2));
J = [term1+term2,term1,term2];

if nargout>1,
  FR1 = [E1;F1;R1;abs(F1-R1)];
  FR2 = [E2;F2;R2;abs(F2-R2)];
end
```

B.1.8 hbdfss_grad7.m

```
function G=hbdfss_grad(u,n,params,E1,E2,lambda)
%HBDFSS_GRAD7 Gradient function for depth from shading and stereo
```

```
%   problem with heirarchical basis functions. Uses true perspective
%   projection.
%
%   G=HBDFSS_GRAD7(U,N,PARAMS,E1,E2,LAMBDA) where U is the disparity
%   map, and N is the number of H-basis levels. E1 and E2 are the input
%   images, LAMBDA is the scalar weighting factor on departure from
%   smoothness. The image parameters PARAMS = [f,b,z0,ps1,qs1,ps2,ps2];

%   Clay M. Thompson 5-23-92
%   Uses u = fb/2z - fb/2z_0 to enhance the numerical stability of the algorithm.
%   Revised to support multi-grid algorithm

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
u0 = f*b/2/z0;
delta = params(12);
area = prod(size(u))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(u);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/2/delta;  % X-derivative
hy = [1;0;-1]/2/delta;  % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/4/delta/delta;
%hxy = [-1 0 1;0 0 0;1 0 -1]/4/delta/delta;
hxy = [-1 1;1 -1]/delta/delta;
hyy = [1 2 1;-2 -4 -2;1 2 1]/4/delta/delta;
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta;

% Compute p,q using numerical derivatives of z.
u = hbasis(u,n);
uu = u + u0;
ux = cfilter2d(u,hx);
uy = cfilter2d(u,hy);
denx = (x.*ux-uu);
deny = (y.*uy-uu);
p = f*ux./denx;
q = f*uy./deny;

% Squared denominators for gradient terms below
denx(:) = denx.*denx;
deny(:) = deny.*deny;

% Compute reflectance map values.
R1  = rmap(p,q,ps1,qs1);
R2  = rmap(p,q,ps2,qs2);
Rp1 = rmapp(p,q,ps1,qs1);
Rp2 = rmapp(p,q,ps2,qs2);
Rq1 = rmapq(p,q,ps1,qs1);
Rq2 = rmapq(p,q,ps2,qs2);

% Compute numerical derivatives of z.
uxx = cfilter2d(u,hxx);
uxy = cfilter2d(u,hxy);
uyy = cfilter2d(u,hyy);

% Compute image projections.
x1 = (x+uu-vx1-1)/delta+1;
x2 = (x-uu-vx2-1)/delta+1;

% Compute 1st difference in the x direction of image arrays.
% Based on the stencil:   -1 1
[m1,n1] = size(E1); [m2,n2] = size(E2);
stencil = [-1 1]/delta;
Ex1 = filter2d(E1,stencil,'resize');
Ex2 = filter2d(E2,stencil,'resize');

```

```
% Determine stereo mapped image derivatives.
Fx1 = interpx(Ex1,floor(x1)); zz = zeros(m1,n1);
out = isnan(Fx1); if any(out(:)), Fx1(out) = zz(out); end
Fx2 = interpx(Ex2,floor(x2)); zz = zeros(m2,n2);
out = isnan(Fx2); if any(out(:)), Fx2(out) = zz(out); end

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,x1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end
F2 = interpx(E2,x2);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end

% Compute Gradient
G = zeros(mz,nz);
ERp1 = (F1-R1).*Rp1; ERq1 = (F1-R1).*Rq1;
ERp2 = (F2-R2).*Rp2; ERq2 = (F2-R2).*Rq2;
G = ((F1-R1).*Fx1 - (F2-R2).*Fx2) ...
    - f*((ERp1+ERp2).*ux./denx + (ERq1+ERq2).*uy./deny);
G(:) = G + cfilter2d((ERp1+ERp2).*uu./denx,-f*hx,'grad');
G(:) = G + cfilter2d((ERq1+ERq2).*uu./deny,-f*hy,'grad');
G(:) = G + cfilter2d(uxx,lambda*hxx,'grad') + ...
    cfilter2d(uxy,2*lambda*hxy,'grad') + ...
    cfilter2d(uyy,lambda*hyy,'grad');

G = hbasis(G*(1/area),n,'trans');
```

B.1.9 hbdfss_cost8b.m

```
function [J,FR1,FR2]=hbdfss_cost(v,n,params,E1,E2,lambda,mu)
%HBDFSS_COST8B Cost function for depth from shading and stereo
%   problem with heirarchical basis functions. Uses true
%   perspective projection and varying albedo
%
%   J=HBDFSS_COST8B(V,N,PARAMS,E1,E2,LAMBDA,MU) where V=[Z,RHO] are
%   the optimization variables (height and albedo), N is the number of
%   H-basis levels.. E1 and E2 are the input images, LAMBDA and MU are
%   scalar weighting factors on departure from smoothness for height
%   and albedo, respectively. The image parameters PARAMS =
%   [f,b,z0,ps1,qs1,ps2,ps2].
%
%   J= [Jtot,Jimage,Jsmooth,Jalbedo].

%   Clay M. Thompson 6-28-92

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2/z0;
delta = params(12);
area = prod(size(E1))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(E1);
z = v(:,1:nz); rho = v(:,nz+[1:nz]);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/(2*delta);     % X-derivative
hy = [1;0;-1]/(2*delta);     % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/(4*delta*delta);
%hxy = [-1 0 1;0 0 0;1 0 -1]/(4*delta*delta);
hxy = [-1 1;1 -1]/(delta*delta);
hyy = [1 2 1;-2 -4 -2;1 2 1]/(4*delta*delta);
del2 = [1 4 1;4 -20 4;1 4 1]/(6*delta*delta);
hr = [0 0 0;0 1 0;0 0 0] - ones(3,3)/9;

% Compute p,q using numerical derivatives of z.
z = hbasis(z,n); rho = hbasis(rho,n);
```

```
zx = cfilter2d(z,hx);
zy = cfilter2d(z,hy);
p = f*zx./(x.*zx+z);
q = f*zy./(y.*zy+z);

% Compute reflectance map values.
R1  = rho.*rmap(p,q,ps1,qs1);
R2  = rho.*rmap(p,q,ps2,qs2);

% Compute numerical derivatives of z.
zxx = cfilter2d(z,hxx);
zxy = cfilter2d(z,hxy);
zyy = cfilter2d(z,hyy);

% Compute disparity.
d = (f*b/2)./z;

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,(x+d-vx1-1)/delta+1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end,
F2 = interpx(E2,(x-d-vx2-1)/delta+1);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end,

%image([F1,F2;R1,R2],[0 1],5)

term1 = (0.5/area)*sum(sum((F1-R1).^2 + (F2-R2).^2));
term2 = (0.5/area)*lambda*(sum(sum(zxx.^2)) + 2*sum(sum(zxy.^2)) + sum(sum(zyy.^2)));
term3 = (0.5/area)*mu*sum(sum(cfilter2d(rho,hr).^2));
%term2 = (0.5/area)*lambda*sum(sum(delsqz.^2));
%disp(sprintf('Terms: %12.5f %12.5f %12.5f',term1,term2,term3));
J = [term1+term2+term3,term1,term2,term3];

if nargout>1,
  FR1 = [E1;F1;R1;abs(F2-R2)];
  FR2 = [E2;F2;R2;abs(F2-R2)];
end
```

B.1.10 hbdfss_grad8b.m

```
function G=hbdfss_grad(v,n,params,E1,E2,lambda,mu)
%HBDFSS_GRAD8B Gradient function for depth from shading and stereo
%   problem with heirarchical basis functions. Uses true perspective
%   projection and varying albedo.
%
%   G=HBDFSS_GRAD8B(V,N,PARAMS,E1,E2,LAMBDA,MU) where V=[Z,RHO] are
%   the optimization variables (height and albedo), N is the number of
%   H-basis levels.. E1 and E2 are the input images, LAMBDA and MU are
%   scalar weighting factors on departure from smoothness for height
%   and albedo, respectively. The image parameters PARAMS =
%   [f,b,z0,ps1,qs1,ps2,ps2].

%   Clay M. Thompson 6-28-92

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
gamma = f*b/2/z0;
delta = params(12)
area = prod(size(E1))*delta*delta;

% Light Source positions
ps1 = params(4); qs1 = params(5);
ps2 = params(6); qs2 = params(7);

% Camera coordinate calibration
vx1 = params(8);  vy1 = params(9);
vx2 = params(10); vy2 = params(11);
vx = (vx1+vx2)/2; vy = (vy1+vy2)/2;

% Spacial coordinates in image.
[mz,nz] = size(E1);
z = v(:,1:nz); rho = v(:,nz+[1:nz]);
[x,y] = domain2d([0:mz-1]*delta+vx+1,[0:nz-1]*delta+vy+1);

% Stencils
hx = [-1 0 1]/2/delta;  % X-derivative
hy = [1;0;-1]/2/delta;  % Y-derivative
hxx = [1 -2 1; 2 -4 2; 1 -2 1]/4/delta/delta;
```

```
%hxy = [-1 0 1;0 0 0;1 0 -1]/4/delta/delta;
hxy = [-1 1;1 -1]/delta/delta;
hyy = [1 2 1;-2 -4 -2;1 2 1]/4/delta/delta;
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta;
hr = [0 0 0;0 1 0;0 0 0] - ones(3,3)/9;

% Compute p,q using numerical derivatives of z.
z = hbasis(z,n); rho = hbasis(rho,n);
zx = cfilter2d(z,hx);
zy = cfilter2d(z,hy);
p = f*zx./(x.*zx+z);
q = f*zy./(y.*zy+z);

% Compute reflectance map values.
RR1  = rmap(p,q,ps1,qs1); R1 = rho.*RR1;
RR2  = rmap(p,q,ps2,qs2); R2 = rho.*RR2;
Rp1 = rho.*rmapp(p,q,ps1,qs1);
Rp2 = rho.*rmapp(p,q,ps2,qs2);
Rq1 = rho.*rmapq(p,q,ps1,qs1);
Rq2 = rho.*rmapq(p,q,ps2,qs2);

% Compute numerical derivatives of z.
zxx = cfilter2d(z,hxx);
zxy = cfilter2d(z,hxy);
zyy = cfilter2d(z,hyy);

% Compute disparity.
d = (f*b/2)./z;
x1 = (x+d-vx1-1)/delta+1;
x2 = (x-d-vx2-1)/delta+1;

% Compute 1st difference in the x direction of image arrays.
% Based on the stencil:   -1 1
[m1,n1] = size(E1); [m2,n2] = size(E2);
stencil = [-1 1]/delta;
Ex1 = filter2d(E1,stencil,'resize');
Ex2 = filter2d(E2,stencil,'resize');

% Determine stereo mapped image derivatives.
Fx1 = interpx(Ex1,floor(x1)); zz = zeros(m1,n1);
out = isnan(Fx1); if any(out(:)), Fx1(out) = zz(out); end
Fx2 = interpx(Ex2,floor(x2)); zz = zeros(m2,n2);
out = isnan(Fx2); if any(out(:)), Fx2(out) = zz(out); end

% Determine stereo mapped images F1,F2.  Set error to zero where F is NaN
F1 = interpx(E1,x1);
out = isnan(F1); if any(out(:)), F1(out) = R1(out); end
F2 = interpx(E2,x2);
out = isnan(F2); if any(out(:)), F2(out) = R2(out); end

% Compute Gradient
FR1 = (F1-R1); FR2 = (F2-R2);
ERp1 = FR1.*Rp1; ERq1 = FR1.*Rq1;
ERp2 = FR2.*Rp2; ERq2 = FR2.*Rq2;
denx = (x.*zx+z).^2;
deny = (y.*zy+z).^2;
Gz = -(f*b/2)*(FR1.*Fx1 - FR2.*Fx2)./(z.^2) + ...
    f*((ERp1+ERp2).*zx./denx + (ERq1+ERq2).*zy./deny);
Gz(:) = Gz - cfilter2d((ERp1+ERp2).*z./denx,-f*hx,'grad');
Gz(:) = Gz - cfilter2d((ERq1+ERq2).*z./deny,-f*hy,'grad');
Gz(:) = Gz + cfilter2d(zxx,lambda*hxx,'grad') + ...
    cfilter2d(zxy,2*lambda*hxy,'grad') + ...
    cfilter2d(zyy,lambda*hyy,'grad');

Grho = cfilter2d(cfilter2d(rho,hr),mu*hr,'grad') - FR1.*RR1 - FR2.*RR2;

Gz = hbasis(Gz*(1/area),n,'trans');
Grho = hbasis(Grho*(1/area),n,'trans');
G = [Gz,Grho];

```

B.1.11 hbsfs_cost.m

```
function [J,FR]=hbsfs_cost(z,levels,params,E,lambda)
%HBSFS  Cost function for shape from shading using hierarchical
%   basis functions.  Based on orthographic projection.
%
%   J=HBSFS_COST(Z,LEVELS,PARAMS,E,LAMBDA) where Z is the depth map,
%   LEVELS is the number of h-basis levels. E is the input image, and
%   LAMBDA is the scalar weighting factor on departure from
%   smoothness. The image parameters are PARAMS = [f,b,z0,ps,qs].

%   Clay M. Thompson 12-16-91

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
delta = params(12);
area = prod(size(z))*delta*delta,

% Light Source positions
ps = params(4); qs = params(5);

% Extract z1 & z2 and transform into nodal basis.
[mz,nz] = size(E); mz = mz+2; nz = nz+2;
z = hbasis(z,levels);

% Stencils
hx = (f/z0)*[-1 0 1]/2/delta;   % X-derivative
hy = (f/z0)*[1;0;-1]/2/delta;   % Y-derivative
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta; % Laplacian

% Compute p,q using numerical derivatives of z.
rows = 2:mz-1; cols = 2:nz-1;
p = filter2d(z,hx,'resize'); p = p(rows,:);
q = filter2d(z,hy,'resize'); q = q(:,cols);

% Compute reflectance map values.
R  = rmap(p,q,ps,qs);

term1 = (0.5/area)*sum(sum((E-R).^2));  % sfs
term2 = (0.5/area)*lambda*sum(sum( filter2d(z,del2,'resize').^2 )); % Smoothness
%disp(sprintf('Terms: %12.5f %12.5f',term1,term2));
J = [term1+term2,term1,term2];

if nargout>1,
  FR = [E;R;abs(E-R)];
end
```

B.1.12 hbsfs_grad.m

```
function G=hbsfs_grad(z,levels,params,E,lambda)
%HBDFSS_GRAD4 Gradient function for shape from shading using
%   hierarchical basis representation.  Based on orthographic
%   projection.
%
%   G=HBSFS_GRAD(Z,LEVELS,PARAMS,E,LAMBDA) where Z is the depth map,
%   LEVELS is the number of h-basis levels. E is the input image, and
%   LAMBDA is the scalar weighting factor on departure from
%   smoothness. The image parameters are PARAMS = [f,b,z0,ps,qs].

%   Clay M. Thompson 12-16-91
%   Revised to support multi-grid scheme.

% Camera constants
f = params(1);
b = params(2);
z0 = params(3);
delta = params(12);
area = prod(size(z))*delta*delta;

% Light Source positions
ps = params(4); qs = params(5);

% Extract z1 & z2 and transform into nodal basis.
[mz,nz] = size(E); mz = mz+2; nz = nz+2;
z = hbasis(z,levels);

% Stencils
hx = (f/z0)*[-1 0 1]/2/delta;   % X-derivative
hy = (f/z0)*[1;0;-1]/2/delta;   % Y-derivative
del2 = [1 4 1;4 -20 4;1 4 1]/6/delta/delta; % Laplacian

% Compute p,q using numerical derivatives of z.
rows = 2:mz-1; cols = 2:nz-1;
p = filter2d(z,hx,'resize'); p = p(rows,:);
q = filter2d(z,hy,'resize'); q = q(:,cols);
del2z = filter2d(z,del2,'resize');

```

```
% Compute reflectance map values.
R  = rmap(p,q,ps,qs);
Rp = rmapp(p,q,ps,qs); Rq = rmapq(p,q,ps,qs);

% Compute error terms

% Now form gradient
G = zeros(mz,nz);
G(rows,:) = filter2d((E-R).*Rp,hx);
G(:,cols) = G(:,cols) + filter2d((E-R).*Rq,hy);
G(:) = G + lambda*filter2d(del2z,del2);

% Return gradient in h-basis coordinates
G = hbasis(G*(1/area),levels,'trans');
```

B.2 Support routines.

B.2.1 rmap.m

```
function R=rmap(p,q,ps,qs)
%RMAP    Reflectance map calculation
%   R=RMAP(P,Q) computes the reflectance map image of the surface with
%   the gradients P and Q (in the x and y direction respectively).
%   P and Q are matrices that contain the gradients over a rectangular
%   grid.
%
%   R=RMAP(P,Q,Ps,Qs) uses the light source direction (Ps,Qs).
%
%   Currently implements: Lambertian reflectance

if nargin==2,
  ps = .1; qs = .1;  % Light source direction
end

[n,m] = size(p);
R = max( (1+ps*p+qs*q) ./ sqrt(1+p.*p+q.*q) ./ sqrt(1+ps*ps+qs*qs) ...
         ,zeros(n,m) );

```

B.2.2 rmapp.m

```
function Rp=rmapp(p,q,ps,qs)
%RMAPP    Reflectance map partial derivative calculation
% Rp=RMAPP(P,Q) computes the reflectance map X partial derivative of
% the surface with gradients P and Q (in the x and y direction
% respectively).  P and Q are matrices that contain the gradients
% over a rectangular grid.
%
% Rp=RMAPP(P,Q,Ps,Qs) uses the light source direction (Ps,Qs).
%
% Currently implements: Lambertian reflectance

if nargin==2,
  ps = .1; qs = .1;  % Light source direction.
end

[n,m] = size(p);
d = ones(n,m) + p.*p + q.*q;
e = ones(n,m) + ps*p + qs*q;
Rp = (ps - e .* p ./ d ) ./ sqrt(d) ./ sqrt(1+ps*ps+qs*qs);

ndx = find(rmap(p,q,ps,qs)==0);
if length(ndx)>0, Rp(ndx) = zeros(length(ndx),1); end

```

B.2.3 rmapq.m

```
function Rq=rmapq(p,q,ps,qs)
%RMAPQ    Reflectance map partial derivative calculation
% Rq=RMAPQ(P,Q) computes the reflectance map Y partial derivative of
% the surface with gradients P and Q (in the x and y direction
% respectively).  P and Q are matrices that contain the gradients
% over a rectangular grid.
%
```

```
% Rq=RMAPQ(P,Q,Ps,Qs) uses the light source direction (Ps,Qs).
%
% Currently implements: Lambertian reflectance

if nargin==2,
  ps = .1; qs = .1;  % Light source direction.
end

[n,m] = size(p);
d = ones(n,m) + p.*p + q.*q;
e = ones(n,m) + ps*p + qs*q;
Rq = (qs - e .* q ./ d) ./ sqrt(d) ./ sqrt(1+ps*ps+qs*qs);

ndx = find(rmap(p,q,ps,qs)==0);
if length(ndx)>0, Rq(ndx) = zeros(length(ndx),1); end

```

B.2.4 conjgrad.m

```
function [x,OPTIONS,J,history]=conjgrad(FUN,x,OPTIONS,GRAD,...
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)
%CONJGRAD Conjugate-gradient optimization.
%
%   X = CONJGRAD('FUN',X0,OPTIONS,'GRAD') finds the minimum of the
%   function 'FUN' with gradient 'GRAD' using a conjugate-gradient
%   optimization algorithm. X0 is an initial condition. OPTIONS is a
%   vector that contains optional information for the optimizer (see
%   FOPTIONS). 'FUN' and 'GRAD' are strings that contain the names of
%   the cost function and gradient M-files, respectively. 'FUN' should
%   return a scalar function value, f=FUN(x). 'GRAD' should return the
%   gradient vector (df/dx), g = GRAD(x).
%
%   Up to ten parameters can be passed to 'FUN' and 'GRAD' using X =
%   CONJGRAD('fun',X0,OPTIONS,'grad',P1,P2,...) so f=FUN(x,P1,P2,...)
%   and g=GRAD(x,P1,P2,...).

%   Note: FUNCTION value is returned in OPTIONS(8) and gradient value
%   is returned in OPTIONS(15).  OPTIONS(19)-History save rate.

% Clay M. Thompson 2-4-91

tol = 1.e-6;          % Minimum allowed alpha

error(nargchk(4,14,nargin));

% Form call strings.
params = [];
for n=5:nargin
  params = [params,',P',int2str(n-4)];
end
if ~any(FUN<48), fcall = [FUN,'(x',params,')']; else fcall = FUN; end
if ~any(GRAD<48), gcall = [GRAD,'(x',params,')']; else gcall = GRAD; end
if ~any(FUN<48),
  LPARAM = [];
  for n=5:nargin,
    LPARAM = [LPARAM,',P',int2str(n-2)];
  end
  SEARCH = ['[eval(''x=P1+x*P2;''),',FUN,'(x',LPARAM,')]'];
else
  SEARCH = ['[eval(''x=P1+x*P2;''),',FUN,']'];
end
linecall = ['lsearch(SEARCH,0,2*alpha,f,GRAD,x,pk',params,')'];

nvars = length(x(:));
[mx,nx] = size(x);

% Initialize parameters
beta = 0;
f = eval(fcall);   % Function value
fold = f;
gk = eval(gcall);  % Gradient
pk = zeros(mx,nx); % Search direction
gknorm = norm(gk(:));
OPTIONS = foptions(OPTIONS);
OPTIONS(10) = OPTIONS(10) + 1;
OPTIONS(11) = OPTIONS(11) + 1;
if OPTIONS(18)==0, alpha=.01; else, alpha = 2*OPTIONS(18); end % Initial guess

if size(gk)~=size(x), error('The size of the gradient and x don''t match.'); end
if OPTIONS(1)>0,
  disp('Fun. Evals --- Value --- Stepsize --- Gradient');
  disp([sprintf('%5.0f %12.3g %12.3g ',OPTIONS(10),f(1),OPTIONS(18)), ...
        sprintf('%12.3g  ',gknorm)]);
end
if OPTIONS(19)>0 & nargout>3,
  history = [OPTIONS(10),f,OPTIONS(10),gknorm];
end

while OPTIONS(10)<OPTIONS(14),
  % Test for convergence.
  if gknorm<OPTIONS(2),
    OPTIONS(8) = f(1); OPTIONS(15) = gknorm;
    disp('Gradient criteria met.'), break
  end
  if OPTIONS(7)==0,
    pk = -gk + beta*pk; % Compute search direction.
  else
    pk = -gk;
  end

  % Do an inexact line search to determine, alpha.
  GRAD = gk(:)'*pk(:);
  if GRAD>0,
    pk = -gk; GRAD = -gk(:)'*gk(:);
    if OPTIONS(1)>0, disp('Redirect search.'),  end,
  end
  [alpha,f,n,how] = eval(linecall); % Line search

  OPTIONS(10) = OPTIONS(10) + n;
  OPTIONS(18) = alpha;

  x = x + alpha*pk;

  if alpha>tol,
    % gkold = gk;                % Needed only with Polak-Ribiere Method
    gk = eval(gcall); OPTIONS(11) = OPTIONS(11) + 1;

    %beta = ((gk-gkold)'*gk)/gknorm;    % Polak-Ribiere Method
    %gknorm = norm(gk(:));

    beta = norm(gk(:))/gknorm;       % Fletcher-Reeves Method
    gknorm = beta*gknorm;
  else
    beta = 0;
    alpha = tol;
    gknorm = norm(gk(:));
    if OPTIONS(1)>0, disp('Reset alpha.'), end
  end

  if OPTIONS(1)>0,
    disp([sprintf('%5.0f %12.3g %12.3g ',OPTIONS(10),f(1),OPTIONS(18)), ...
          sprintf('%12.3g ',gknorm),how]);
  end

  if OPTIONS(19)>0 & nargout>3,
    if OPTIONS(10)>=history(length(history(:,1)),1)+OPTIONS(19),
      history = [history;[OPTIONS(10),f,OPTIONS(18),gknorm]];
    end
  end

  % Test for convergence on relative change in F.
  if abs((fold(1)-f(1))/(fold(1)+eps))<OPTIONS(3),
    OPTIONS(8) = f(1); OPTIONS(15) = gknorm;
    disp('Relative function change criteria met.'), break
  end
  fold = f;
end
OPTIONS(8) = f(1);
OPTIONS(15) = gknorm;
J = f;
```

B.2.5 lsearch.m

```
function [alpha,f,n,how] = lsearch(FUN,x0,alpha1,f0,g0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12)
%LSEARCH Inexact line search.
%
%   [ALPHA,F,N] = LSEARCH('FUN',X,ALPHA,F,G) performs an inexact
%   line search of the 1-D function 'FUN'. 'FUN' is a string variable
%   that defines the name of an M-file function to be minimized. The
%   function should return a scalar value, f=FUN(x). X is the initial
%   starting point. F and G are the value and gradient of the function
%   at the initial point X. G must be negative (i.e., downhill). ALPHA
%    is an initial guess for the step length (i.e. X1 = X + ALPHA).
%    LSEARCH returns the step length ALPHA, the function value F, and
%    the number of function evaluations N.
%
%    Up to ten additianal parameters can be passed to 'fun' using
%    [ALPHA,F,N] = LSEARCH('FUN',X,ALPHA,F,G,P1,P2,P3,...) in which case
%    the function is called using FUN(x,P1,P2,P3,...).

%    Clay M. Thompson 4-11-91

showSteps = 0;
showPlot = 0;

tol = 1e-7; % Minimum tolerable value for alpha

% Check for valid initial condition.
if g0>0, error('Initial gradient is not downhill.'); end

% Form call string
params = '(x';
for n=6:nargin
  params = [params,',P',int2str(n-5)];
end
if ~any(FUN<48), fcall = [FUN,params,')']; else fcall = FUN; end

how = '';
n = 0;

if showSteps, disp('---Alpha ------ Function Value'), end
if showPlot,
  hold off, plot([0 5*alpha1],[f0(1) f0(1)+5*alpha1*g0],'b-'), hold on
  plot(0,f0(1),'bo'),
  s=-alpha1:alpha1/10:5*alpha1;
end

while n<9,
  % Evaluate function at x+alpha1.
  x = x0+alpha1;
  f1 = eval(fcall); n = n+1;
  if showSteps, disp(['1: ',num2str(alpha1),'  ',num2str(f1),'  ',how]), end
  if showPlot, plot(alpha1,f1(1),'bo'), end

  % Fit quadratic function to the points x0,x1.  f = a*x^2 + b*x + c.
  c = f0(1); b = g0; a = (f1(1)-b*alpha1-c)/(alpha1*alpha1);

  if showPlot, plot(s,polyval([a,b,c],s),'r-'), end % Fitted curve

  % Determine next jump based on the quadratic fit.
  if (a>eps),   % Jump forward but not too much.
    alphaq = -b/(2*a);
    if (alphaq<5*alpha1) & (alphaq~=alpha1),
      alpha2 = alphaq;
      how = [how,'quadratic fit; '];
    else
      alpha2 = 5*alpha1;
      how = [how,'limited jump; '];
    end

  elseif (a<eps),   % Optimum is maximum, jump forward.
    alpha2 = 5*alpha1;
    how = [how,'Increase step size (q); '];

  end

  % Evaluate function at x+alpha2.
  x = x0 + alpha2;
  f2 = eval(fcall); n = n+1;
  if showSteps, disp(['2: ',num2str(alpha2),'  ',num2str(f2(1)),'  ',how]), end
  if showPlot, plot(alpha2,f2(1),'x'), end  % Chosen point

  % Check for adequate solution
%  if (f2(1)<min(f0(1),f1(1))) & (alpha2<alpha1),
%    alpha = alpha2;
%    f = f2;
%    break      % Normal exit
%  end

  % Try to fit a cubic function,  f = a*x^3 + b*x^2 + c*x + d.
  d = f0(1); c = g0;
  alphamat = [alpha1.^3 alpha1.^2; alpha2.^3 alpha2.^2];
  if rcond(alphamat)<eps, % Cubic solution is invalid, use quadratic
     alpha = alpha2;
    f = f2;
    return   % Normal exit
  end
  ab = alphamat\[f1(1)-c*alpha1-d;f2(1)-c*alpha2-d];
  a = ab(1); b = ab(2);

  if showPlot, plot(s,polyval([a,b,c,d],s),'g-'), end % Fitted curve

  % Check descriminent
  del = b*b-3*a*c;
  if (del>0) & abs(a)>eps,  % Solve for roots.
    alpha3 = (-b+sqrt(del))/(3*a);
  elseif abs(a)<eps     % Function looks VERY quadratic.
    alpha3 = alpha2;
  else
    alpha3 = inf;   % Flag invalid fit.
  end

  if alpha3<0,      % Reduce step size and start over.
    alpha1 = min(alpha1,alpha2)/2;
    how = [how,'Reduce step size (c); '];
    if showSteps, disp(['3: ',num2str(alpha3),'        ',how]), end

  elseif alpha3==inf,  % Increase step size and start over.
    alpha1 = 5*max(alpha1,alpha2);
    how = [how,'Invalid cubic fit; '];
    if showSteps, disp(['3: ',num2str(alpha3),'        ',how]), end

  else
    if (alpha3-alpha2)/alpha1 < .01, % Extra function evaluation not necessary
      f3 = inf;
    else
      how = [how,'Cubic fit; '];
      % Evaluate function at x+alpha3.
      x = x0 + alpha3;
      f3 = eval(fcall); n = n+1;
      if showSteps, disp(['3: ',num2str(alpha3),'  ',num2str(f3(1)),'  ',how]), end
      if showPlot, plot(alpha3,f3(1),'x'), end  % Chosen point
    end

    % Check if minimum is bracketed.
    ftest = [f1(1),f2(1),f3(1)];
    i = find(min(ftest)==ftest);
    alpha = eval(['alpha',int2str(i(1))]);
    f = eval(['f',int2str(i(1))]);

    if (f(1)<f0(1)) & (abs(alpha)>tol),
      return;             % Normal exit

    elseif abs(alpha)<tol, % GRAD may not be accurate.
      x = x0 + tol;
      f = eval(fcall); n = n+1;
      alpha = 10*rand(1)*tol;
      how = [how,'Random Jump; '];
      return

    else
%if alpha==min([alpha1,alpha2,alpha3]),
      alpha1 = alpha/2;
      how = [how,'Reduce step size (b); '];

%    else
%      alpha1 = alpha;
%      how = [how,'Choose smallest; '];
    end

  end

end % while

% Abnormal exit.  No improvement in cost.  Use min value for alpha
f = f0;
alpha = 0;
disp('WARNING: Exceeded 9 line search attempts.')
```

B.2.6 filter2d.m

```
function x = filter2d(a,stencil,resize)
%FILTER2D Two dimensional computational stencil filtering.
%
%   X = FILTER2D(A,STENCIL) returns X which is the result of
%   applying the computational molecule STENCIL to the matrix A.
```

```
%
%    If SIZE(A) is ma-by-na and SIZE(STENCIL) is ms-by-ns then
%    SIZE(X) is (ma+ms-1)-by-(na+ns-1).
%
%    FILTER2D(A,STENCIL) is the same as CONV2(A,ROT90(STENCIL,2)).

%    Clay M. Thompson 1-15-91

[ms,ns] = size(stencil);
[ma,na] = size(a);

% This calculation is the same as conv2(a,rot90(stencil,2))
% but is faster.
if 0,
  x = zeros(ms+ma-1,ns+na-1);
  for i=1:ms
    for j=1:ns
      w = stencil(ms-i+1,ns-j+1);
      if w~=0,
        x = x + [zeros(i-1,j-1),zeros(i-1,na),zeros(i-1,ns-j);
                 zeros(ma,j-1),w*a,zeros(ma,ns-j);
                 zeros(ms-i,j-1),zeros(ms-i,na),zeros(ms-i,ns-j)];
      end
    end
  end

else
  x = conv2(a,rot90(stencil,2));
end

if nargin==3, % Return the central (valid) part.
  rows = ms-1 + [1:ma-ms+1];
  cols = ns-1 + [1:na-ns+1];
  x = x(rows,cols);
end
```

B.2.7 cfilter2d.m

```
function c=cfilter2d(a,b,grad)
%CFILTER2D  Filter bicubic approximation of array.
%
%    C = CFILTER2D(A,B) applies the filter B to the array A where
%    A is interpolated using bicubic interpolation.
%
%    C = CFILTER2D(A,B,'grad') returns the convolution of B with A
%    for a gradient calculation.
%
% The cubic approximation is used extrapolate a 1 pixel border
% around the array A.
%
%    See also: FILTER2D.

%    Clay M. Thompson 4-30-92

[ma,na] = size(a);  % Size of array.
[mb,nb] = size(b);  % Filter size.

if nargin==2,
if nb>1,
  a = [3*a(:,1)-3*a(:,2)+a(:,3),a,3*a(:,na)-3*a(:,na-1)+a(:,na-2)];
end
if mb>1,
  a = [3*a(1,:)-3*a(2,:)+a(3,:);a;3*a(ma,:)-3*a(ma-1,:)+a(ma-2,:)];
end
  c = filter2d(a,b,'resize');
else
  c = filter2d(a,b);
  if nb>1,
    c(:,2:4) = c(:,2:4) + c(:,1)*[3 -3 1];
    c(:,na+nb-4:na+nb-2) = c(:,na+nb-4:na+nb-2) + c(:,na+nb-1)*[1 -3 3];
    c = c(:,2:nb+na-2);
  end
  if mb>1,
    c(2:4,:) = c(2:4,:) + [3;-3;1]*c(1,:);
    c(ma+mb-4:ma+mb-2,:) = c(ma+mb-4:ma+mb-2,:) + [1;-3;3]*c(ma+mb-1,:);
    c = c(2:mb+ma-2,:);
  end
end
```

B.2.8 hbasis.m

```
function y = hbasis(x,n,oper)
%HBASIS Map from hierarchical basis representation to nodal
%   representation.
%
%   Y = HBASIS(X,N) or Y = HBASIS(X,N,'nodal') maps X from a
%   hierarchical basis with N levels to the nodal basis.
%
%   Y = HBASIS(X,N,'trans') maps from the nodal basis to the
%   hierarchical basis with N levels using the adjoint map.
%
%   Y = HBASIS(X,N,'inv') maps from the nodal basis to the
%   hierarchical basis with N levels.
%
%   Y = HBASIS(X,N,'tinv') maps from the nodal basis to the
%   hierarchical basis with N levels using the adjoint map.

%   Reference: "Fast Surface Interpolation Using Heirarchical Basis
%   Functions", Richard Szeliski, IEEE PAMI, Vol 12, No. 6, June 1990.

%   Clay M. Thompson  4-2-91
%   Revised 6-10-91 by CMT

error(nargchk(2,3,nargin));

if nargin<3, code = 'no'; else code = oper(1:2); end

% Check to make sure the size of x is compatible with n levels.
[mm,nn] = size(x);
incr = 2^(n-1);
if mm<incr | nn<incr,
  error(['Each dimension of X must be larger than ',int2str(incr), ...
  ' for ',int2str(n),' levels.']);
end

if code=='no', % oper=='nodal'
  y = zeros(mm,nn);
  rows = 1:incr:mm;  cols = 1:incr:nn;
  y(rows,cols) = x(rows,cols);
  for level=(n-1):-1:1,
    xrows = 2*rem(mm-1,incr) >= incr; xcols = 2*rem(nn-1,incr) >= incr;
    incr = incr/2;
    row2 = 1:incr:mm;  col2 = 1:incr:nn;
    y(row2,col2) = hb(y(rows,cols),xrows,xcols) + x(row2,col2);
    y(rows,cols) = y(rows,cols) - x(rows,cols);     % Remove extra term
    rows = row2; cols = col2;
  end

elseif code=='tr', % oper=='trans',
  y = zeros(mm,nn);
  incr = 1;
  rows = 1:incr:mm;  cols = 1:incr:nn;
  y(rows,cols) = x(rows,cols);
  for level=1:(n-1),
    incr = 2*incr;
    row2 = 1:incr:mm;  col2 = 1:incr:nn;
    y(row2,col2) = y(row2,col2) + hbt(y(rows,cols));
    rows = row2; cols = col2;
  end

elseif code=='in', % oper=='inv',
  incr = 1;
  y = zeros(mm,nn);
  rows = 1:incr:mm;  cols = 1:incr:nn;
  y(rows,cols) = x(rows,cols);
  for level=1:(n-1),
    incr = incr*2;
    row2 = 1:incr:mm;  col2 = 1:incr:nn;
    xrows = 2*rem(mm-1,incr) >= incr; xcols = 2*rem(nn-1,incr) >= incr;
    y(rows,cols) = y(rows,cols) - hb(y(row2,col2),xrows,xcols);
    y(row2,col2) = y(row2,col2) + x(row2,col2); % Add back extra term
    rows = row2; cols = col2;
  end

elseif code=='ti', % oper=='tinv',
  y = zeros(mm,nn);
  rows = 1:incr:mm;  cols = 1:incr:nn;
  y = x;
  for level=(n-1):-1:1,
    incr = incr/2;
    row2 = 1:incr:mm;  col2 = 1:incr:nn;
    y(rows,cols) = y(rows,cols) - hbt(y(row2,col2));
    rows = row2; cols = col2;
  end

else
  error(['The operation ',oper,' is invalid.']);
end


```

B.2.9 hb.m

```
function y=hb(x,xtrarows,xtracols)
%HB Interpolate a 2-D function on a rectangular grid using
%   Hierarchical Basis functions.
%
%   Y = HB(X) returns X interpolated to the next level. If X is
%   m-by-n then Y is (2*m-1)-by-(2*n-1).

%   Clay M. Thompson 4-2-91

colmask = [.5 .5];
rowmask = [.5;.5];

if nargin<2, xtrarows = 0; end
if nargin<3, xtracols = 0; end

[m,n] = size(x);
if xtrarows | xtracols,
  x = [x,zeros(m,xtraco  `;zeros(xtrarows,n),zeros(xtrarows,xtracols)];
end

[m,n] = size(x);
mm = 2*m-1; nn = 2*n-1;
oddrows = 1:2:mm;
evenrows = 2:2:mm-1;
oddcols = 1:2:nn;
evencols = 2:2:nn-1;

y = zeros(mm,nn);
y(oddrows,oddcols) = x;
y(oddrows,evencols) = filter2d(x,colmask,'resize');
y(evenrows,:) = filter2d(y(oddrows,  ,rowmask,'resize');

if xtrarows | xtracols,
  y = y(1:mm-xtrarows,1:nn-xtracols);
end
```

B.2.10 hbt.m

```
function y = hbt(x)
%HBT    Decimate a 2-D function on a rectangular grid using Hierarchical Basis
%       functions.
%
%       Y = HBT(X) returns X decimated to the next coarsest level.
%       If X is m-by-n then Y is ((m+1)/2)-by-((n+1)/2).
%
%       Note (m+1) and (n+1) must be divisible by 2.

%       Clay M. Thompson  4-2-91

[m,n] = size(x);
rows = [1:2:m]; cols = [1:2:n];
a = [0,zeros(1,n),0;zeros(m,1),x,zeros(m,1);0,zeros(1,n),0];

y =   a(rows,  cols) + 2*a(rows,  cols+1) +   a(rows,  cols+2) + ...
    2*a(rows+1,cols)                      + 2*a(rows+1,cols+2) + ...
      a(rows+2,cols) + 2*a(rows+2,cols+1) +   a(rows+2,cols+2);
y = y/4;


```

B.2.11 interpx.m

```
function F=interpx(E,x)
%INTERPX Linear interpolation in the x-direction.
%
%       F = INTERPX(E,X) returns a matrix F containing the values of E
%       at the points X. The matrix X must have the same number of rows as
%       E. The values in the matrix X must be between 1 and N, where N is
%       the number of columns of E. The value NaN will be returned where
%       this is not the case.

% Clay M. Thompson  10-17-90

[m,n] = size(E);
[mx,nx] = size(x);
if mx~=m, error('X must have the same number of rows as E.'); end

% Compute nearest x position, xlow.
xlow = floor(x);
out = (x==n);
nout = sum(out(:));
if nout>0, xlow(out) = (n-1)*ones(nout,1); end

% Check for out of range values of x and set to 1
out = (x<1)|(x>n);
nout = sum(out(:));
if nout>0, x(out) = ones(nout,1); xlow(out) = ones(nout,1); end

% Determine index into matrix elements
% Note: y=[1:m]'*ones(1:nx);
elem = [1:m]'*ones(1:nx) + (xlow-1)*m;
F = E(elem) + (x-xlow) .* (E(elem+m)-E(elem));

% Set values of F where x is out of range to NaN.
if any(out(:)), F(out) = NaN*ones(nout,1); end

```

B.2.12 domain2d.m

```
function [x,y] = domain2d(x,y)
%DOMAIN2D  Generate X and Y arrays for 3-d plots.
%    [XX,YY] = DOMAIN2D(X,Y)  transforms the domain specified by vectors
%    X and Y into arrays XX and YY that can be used for the evaluation
%    of functions of two variables with 3-d mesh plots.  For example,
%    to evaluate the function  x*exp(-x^2-y^2) over the range  -2 < x < 2
%    -2 < y < 2,
%
%            [x,y] = meshdom(-2:.2:2, -2:.2:2);
%            z = x .* exp(-x.^2 - y.^2);
%            mesh(z)

%    J.N. Little 12-2-85
%    Revised 20-May-90, LS.
%    Copyright (c) 1985, 1986, 1990 by the MathWorks, Inc.

nx = length(x);
ny = length(y);
x = x(:).';  % make sure x is a row vector
x = x(ones(ny, 1),:);
y = y(ny:-1:1); y = y(:);   % make sure y is a column vector
y = y(:,ones(1, nx));

```

B.2.13 icubic.m

```
function F=icubic(x,y,u)
%ICUBIC Cubic Interpolation of a 1-D function.
%
%    F=ICUBIC(Y,XI) returns the value of the 1-D function Y at the
%    points XI using cubic interpolation. length(F)=length(XI). XI is
%    an index into the vector Y. Y is the value of the function
%    evaluated uniformly on a interval. If Y is a matrix, then
%    the interpolation is performed for each column of Y.
%
%    If Y is of length N then XI must contain values between 1 and N.
%    The value NaN is returned if
%    this is not the case.
%
%    F = ICUBIC(X,Y,XI) uses the vector X to specify the coordinates
%    of the underlying interval. X must be equally spaced and
%    monotonic. XI must lie within the coordinates in X.
%
%    See also: ILINEAR.

%    Clay M. Thompson 7-4-91

%    Based on "Cubic Convolution Interpolation for Digital Image
%    Processing", Robert G. Keys, IEEE Trans. on Acoustics, Speech, and
%    Signal Processing, Vol. 29, No. 6, Dec. 1981, pp. 1153-1160.

if nargin==2,  % No X specified
```

```
  u = y; y = x;
% Check for vector problem.  If so, make everything a column vector.
  if min(size(y))==1, y = y(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);

elseif nargin==3, % X specified.
  % Check for vector problem.  If so, make everything a column vector.
  if min(size(y))==1, y = y(:); end
  if min(size(x))==1, x = x(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);
  % Scale and shift u to be indices into Y.
  if (min(size(x))~=1), error('X must be a vector.'); end
  x = x(:);
  [m,n] = size(x);
  if m ~= nrows,
    error('The length of X must match the number of rows of Y.');
  end
  u = 1 + (u-x(1))*((nrows-1)/(x(m)-x(1)));

else
  error('Wrong number of input arguments.');
end

if nrows<3, error('Y must have at least 3 rows.'); end
[m,n] = size(u);
if n==1, u = u*ones(1,ncols); [m,n] = size(u); end % Expand u
if n~=ncols, error('The number of columns in XI and Y must match.'); end

% Check for out of range values of u and set to 1
uout = find((u<1)|(u>nrows));
nuout = length(uout);
if nuout>0, u(uout) = ones(nuout,1); end

% Interpolation parameters
s = (u - floor(u));
u = floor(u);
d = find(u==nrows); if length(d)>0, u(d) = u(d)-1; s(d) = s(d)+1; end

% Expand y so interpolation is valid at the boundary.
y = [3*y(1,:)-3*y(2,:)+y(3,:);y;3*y(nrows,:)-3*y(nrows-1,:)+y(nrows-2,:)];
nrows = nrows + 2;

% Now interpolate using computationally efficient algorithm.
s2 = s.*s; s3 = s.*s2;
ndx = u+ones(m,1)*[0:n-1]*nrows;
F     = y(ndx).*(-s3+2*s2-s) + y(ndx+1).*(3*s3-5*s2+2) + ...
    y(ndx+2).*(-3*s3+4*s2+s) + y(ndx+3).*(s3-s2);
F = F/2;

% Now set out of range values to NaN.
if nuout>0, F(uout) = NaN*ones(nuout,1); end

```

B.2.14 dcubicx.m

```
function G=dcubicx(x,y,u)
%DCUBICX Derivative of Cubic Interpolation of a 1-D function w.r.t. x..
%
%   G = DCUBICX(Y,XI) returns the derivative of the 1-D function Y=f(x)
%   at the points XI using cubic interpolation. length(G)=length(XI).
%   XI specifies the points originally used for the interpolation.
%
%   If Y is of length N then XI must contain values between 1 and N.
%   The value NaN is returned if this is not the case.
%
%   G = DCUBICX(X,Y,XI) uses the vector X to specify the coordinates
%   for Y as for ICUBIC.
%
%   See also: ICUBIC.

%   Clay M. Thompson 7-18-91

%   Based on "Cubic Convolution Interpolation for Digital Image
%   Processing", Robert G. Keys, IEEE Trans. on Acoustics, Speech, and
%   Signal Processing, Vol. 29, No. 6, Dec. 1981, pp. 1153-1160.

if nargin==2,   % No X specified.
  u = y; y = x;
% Check for vector problem.  If so, make everything a column vector.
```

```
  if min(size(y))==1, y = y(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);

elseif nargin==3, % X specified.
  % Check for vector problem.  If so, make everything a column vector.
  if min(size(y))==1, y = y(:); end
  if min(size(x))==1, x = x(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);
  % Scale and shift u to be indices into Y.
  if (min(size(x))~=1), error('X must be a vector.'); end
  x = x(:);
  [m,n] = size(x);
  if m ~= nrows,
    error('The length of X must match the number of rows of Y.');
  end
  u = 1 + (u-x(1))*((nrows-1)/(x(m)-x(1)));

else
  error('Wrong number of input arguments.');
end

if nrows<3, error('Y must have at least 3 rows.'); end
[m,n] = size(u);
if n==1, u = u*ones(1,ncols); [m,n] = size(u); end  % Expand u
if n~=ncols, error('The number of columns in X1 and Y must match.'); end

% Check for out of range values of u and set to 1
uout = find((u<1)|(u>nrows));
nuout = length(uout);
if nuout>0, u(uout) = ones(nuout,1); end

% Interpolation parameters
s = (u - floor(u));
u = floor(u);
d = find(u==nrows); if length(d)>0, u(d) = u(d)-1; s(d) = s(d)+1; end

% Expand y so interpolation is valid at the boundary.
y = [3*y(1,:)-3*y(2,:)+y(3,:);y;3*y(nrows,:)-3*y(nrows-1,:)+y(nrows-2,:)];
nrows = nrows + 2;

% Now interpolate using computationally efficient algorithm.
s2 = s.*s;
ndx = u+ones(m,1)*[0:n-1]*nrows;
G     = y(ndx).*(-3*s2+4*s-1) + y(ndx+1).*(9*s2-10*s) + ...
     y(ndx+2).*(-9*s2+8*s+1) + y(ndx+3).*(3*s2-2*s);
G = G/2;

% Now set out of range values to NaN.
if nuout>0, G(uout) = NaN*ones(nuout,1); end

```

B.2.15 dcubicz.m

```
function g=dcubicy(x,y,u,dJdf)
%DCUBIC Derivative of 1-D cubic interpolation w.r.t. Y.
%
%   G = DCUBICY(Y,X1,dJdF) computes the derivative of the cost
%   function J(F) with respect to the underlying variables Y. The
%   matrix dJdF is the derivative of the cost function with respect to
%   the interpolated value F = ICUBIC(Y,X1). X1 specifies the points
%   originally used for the interpolation.
%
%   G = DCUBICY(X,Y,X1,dJdF) uses the vector X to specify the
%   coordinates for Y as for ICUBIC.
%
%   See also: ICUBIC.

%   Clay M. Thompson 7-12-91

if nargin==3,   % No X specified.
  dJdf = u; u = y; y = x;
  % Check for vector problem.  If so, make everything a column vector.
  if min(size(y))==1, y = y(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);

elseif nargin==4, % X specified.
  % Check for vector problem.  If so, make everything a column vector.
  if min(size(y))==1, y = y(:), end
  if min(size(x))==1, x = x(:); end
  if min(size(u))==1, u = u(:); end
  [nrows,ncols] = size(y);
  % Scale and shift u to be indices into Y.
  if (min(size(x))~=1), error('X must be a vector.'); end
  x = x(:);
  [m,n] = size(x);
  if m ~= nrows,
    error('The length of X must match the number of rows of Y.');
  end
  u = 1 + (u-x(1))*((nrows-1)/(x(m)-x(1)));

else
  error('Wrong number of input arguments.');
end

if nrows<3, error('Y must have at least 3 rows.'); end
[m,n] = size(u);
if n==1, u = u*ones(1,ncols); [m,n] = size(u); end  % Expand u
if any(size(u)~=size(dJdf)), error('dJdF and X1 must be the same size.'); end
if n~=ncols, error('The number of columns in X1 and Y must match.'); end
if m<2,  % Expand u (so sums work) with an out of range value.
  u = [u;zeros(1,n)]; [m,n] = size(u);
  dJdf = [dJdf;zeros(1,n)];
end

% Check for out of range values of u and set to 1
uout = find((u<1)|(u>nrows));
nuout = length(uout);
if nuout>0, u(uout) = ones(nuout,1); end

% Interpolation parameters
s = (u - floor(u));
u = floor(u);
d = find(u==nrows); if length(d)>0, u(d) = u(d)-1; s(d) = s(d)+1; end

% Compute terms for gradient
s2 = s.*s; s3 = s.*s2;
t0 = dJdf.*(-s3+2*s2-s);
t1 = dJdf.*(3*s3-5*s2+2);
t2 = dJdf.*(-3*s3+4*s2+s);
t3 = dJdf.*(s3-s2);
clear s s2 s3

% Set out of range terms to zero.
if nuout>0,
  t0(uout) = zeros(nuout,1);
  t1(uout) = zeros(nuout,1);
  t2(uout) = zeros(nuout,1);
  t3(uout) = zeros(nuout,1);
  dJdf(uout) = zeros(nuout,1);
end

% Form sums for each z value.
t0sum = zeros(nrows,ncols);
t1sum = zeros(nrows,ncols);
t2sum = zeros(nrows,ncols);
t3sum = zeros(nrows,ncols);
for k=1:nrows,
  elem = find(u==k);
  temp = zeros(m,n);
  temp(elem) = t0(elem); t0sum(k,:) = sum(temp);
  temp(elem) = t1(elem); t1sum(k,:) = sum(temp);
  temp(elem) = t2(elem); t2sum(k,:) = sum(temp);
  temp(elem) = t3(elem); t3sum(k,:) = sum(temp);
end
clear t0 t1 t2 t3

% Add terms from boundary conditions
t1sum(1,:) = t1sum(1,:) + 3*t0sum(1,:);
t2sum(1,:) = t2sum(1,:) - 3*t0sum(1,:);
t3sum(1,:) = t3sum(1,:) + t0sum(1,:);

t2sum(m-1,:) = t2sum(m-1,:) + 3*t3sum(m-1,:);
t1sum(m-1,:) = t1sum(m-1,:) - 3*t3sum(m-1,:);
t0sum(m-1,:) = t0sum(m-1,:) + t3sum(m-1,:);

% Now combine to form gradient.
g = zeros(nrows,ncols);
g(1,:) = t0sum(2,:) + t1sum(1,:);
g(2,:) = t0sum(3,:) + t1sum(2,:) + t2sum(1,:);
for k=3:nrows-1,
  g(k,:) = t0sum(k+1,:) + t1sum(k,:) + t2sum(k-1,:) + t3sum(k-2,:);
end
```

```
g(nrows,:) = t1sum(nrows,:) + t2sum(nrows-1,:) + t3sum(nrows-2,:);

g = g/2;



```

Appendix C

Test Surface Descriptions

This appendix contains information on how to create the crater and hill test surfaces and associated test images.

C.1 Crater Surface

The crater on the flat plane is formed by intersecting two spheres, one with radius 10 and the other with radius 9.48, with a flat plane. The routines **makepair4** and **crater_depth** were used to create the crater images.

C.1.1 makepair4.m

```
% Make stereo pair (this is the best routine)
clear
DEPTH = 'crater_depth'
%DEPTH = 'sphere_depth'
%GRAD = 'crater_grad'
n = 65; % The size of image.
mu = 1; % Optimization parameter

b = 500;
f = -1000; % We are working in the same coordinates as the surface for x & y.

% Camera coordinate calibration
z0 = eval([DEPTH,'(0,0)'])
v0 = 0;
v1 = v0 + b*f/z0/2
v2 = v0 - b*f/z0/2

gamma = f*b/2;
delta = z0/f;
if 1 % Case a {Hard case}
  ps1 = .1; qs1 = .1;
  ps2 = -.1; qs2 = .1;
  prefix = ['cr',int2str(n),'a'];
else % Case b {Easy case}
  ps1 = .2; qs1 = -.5;
  ps2 = -.3; qs2 = .1;
  prefix = ['cr',int2str(n),'b'];
end
%ps1 = .5; qs1 = -.6;
%ps2 = .5; qs2 = -.6;
%ps1 = 0; qs1 = 0;

[x,y] = domain2d(-12:24/(n+1):12,-12:24/(n+1):12 );

% Image parameters
```

```
deltax = x(1,2)-x(1,1);
params = [f/deltax      % Focal length for unity spacing grid
          b         % baseline distance
          z0            % Calibration depth
          ps1;qs1;ps2;qs2   % Light positions
          (x(2,2)+v1)/deltax-1  % x-calibration as offset from E(1,1).
          y(2,2)/deltax-n   % y-calibration as offset from E(1,1).
          (x(2,2)+v2)/deltax-1  % x-calibration as offset from E(1,1).
          y(2,2)/deltax-n   % y-calibration as offset from E(1,1).
          1 % Grid spacing
];
del = z0*(max(x(:))-min(x(:)))/f/(n+1);

z = eval([DEPTH,'(x,y)']);
eqmesh(z,del)
title('True Surface')

% Determine depth function for left image (1)

err = 1; m = 0;
z1 = z0*mones(x);
gamma/z0
while (err> 1.e-5)&(m<25),
  ztemp = eval([DEPTH,'(x+v1-gamma./z1-v0,y)']);
  err = sum( (ztemp(:)-z1(:)).^2 )
  z1 = z1 + mu*(ztemp-z1);
  m = m + 1;
end
eqmesh(z1,del)
title('Left Surface')

% determine depth function for right image (2)
err = 1; m = 0;
z2 = z0*mones(x);
while (err>1.e-5)&(m<25),
  ztemp = eval([DEPTH,'(x+v2+gamma./z2-v0,y)']);
  err = sum( (ztemp(:)-z2(:)).^2 )
  z2 = z2 + mu*(ztemp-z2);
  m = m + 1;
end
eqmesh(z2,del)
title('Right Surface')

hx = [-1 0 1]/2;
hy = [1;0;-1]/2;
rows = 2:n+1;
cols = 2:n+1;

% Right Image
zx = filter2d(z1,hx,'resize')/deltax;
zy = filter2d(z1,hy,'resize')/deltax;
p1 = f*zx(rows,:)./((x(rows,cols)+v1).*zx(rows,:)+z1(rows,cols));
q1 = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z1(rows,cols));
E1 = rmap(p1,q1,ps1,qs1);

% Left Image
zx = filter2d(z2,hx,'resize')/deltax;
zy = filter2d(z2,hy,'resize')/deltax;
p2 = f*zx(rows,:)./((x(rows,cols)+v2).*zx(rows,:)+z2(rows,cols));
q2 = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z2(rows,cols));
E2 = rmap(p2,q2,ps2,qs2);

% Center Global coordinate system
zx = filter2d(z,hx,'resize')/deltax;
zy = filter2d(z,hy,'resize')/deltax;
p = f*zx(rows,:)./(x(rows,cols).*zx(rows,:)+z(rows,cols));
q = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z(rows,cols));
E = rmap(p,q,ps2,qs2);

clg,
subplot(121), eqmesh(z1,del),
subplot(122), eqmesh(z2,del)
subplot(111), title('Image contours')

mimage([E1,zeros(n,1),E2],[0 1])
clear p1 q1 p2 q2 zx zy p q E gamma err m n ztemp
clear ps1 ps2 qs1 qs2 z0 b0 b f delta hx hy rows cols v1 v2
clear mu v0 x y deltax
```

C.1.2 crater_depth.m

```
function [z,rho]=depth(x,y)
%CRATER_DEPTH  Depth function for crater on a flat plane.
%
%   Z = CRATER_DEPTH(X,Y).  X and Y must be of the same size.
%
%   Crater is defined between -10 <= x,y <= 10.

z0 = -1000;
r1 = 10; r2 = 9.48;
d1 = find(x.*x + y.*y <= 0.75*r1*r1);
d2 = find(x.*x + y.*y <= 0.75*r2*r2);

[m,n] = size(x);
z = z0*ones(m,n);
z(d1) = z(d1) + sqrt(r1*r1 - x(d1).*x(d1) - y(d1).*y(d1))-5;
z(d2) = min(z(d2), z0+12-sqrt(r2*r2 - x(d2).*x(d2) - y(d2).*y(d2)));

%z = z0 + (z-z0)/2; % Reduce height by two

if nargout>1,
  % Albedo (rho)
  rho = mones(z);
  d = find(abs(x-y)<3);
  rho(d) = 0.7*mones(d); % Dark strip across diagonal
end
```

C.2 Hill Surface

A fractal based method was used to create the underlying data matrix for the hill surface. Given this underlying data matrix the hill surface is formed using bicubic interpolation to define the surface points. The data for the hill is shown in the matrix (DATA_MATRIX) below.

$$-1000 + \begin{bmatrix} 0.7210 & 1.0934 & 1.1456 & 0.7672 & 0.5582 & 0.6868 & 1.0628 \\ 0.9350 & 0.4301 & 1.2280 & 0.4993 & 0.5555 & 0.8574 & 0.3552 \\ 0.8864 & 1.0791 & 0.2720 & 0.5742 & 0.6650 & 0.3307 & 0.2967 \\ 0.7894 & 0.2520 & 0.5136 & 0.2594 & 1.1026 & 1.0038 & 0.4796 \\ 0.8690 & 0.4103 & 0.3271 & 0.5276 & 1.0706 & 1.1861 & 0.4380 \\ 1.1673 & 1.1548 & 0.3879 & 0.2889 & 0.3674 & 0.8726 & 0.7950 \\ 1.3493 & 0.4233 & 0.3691 & 0.2568 & 0.4342 & 0.6595 & 0.5902 \end{bmatrix}. \qquad (C.1)$$

The routines **makepair_data** and **data_depth** were used to create the hill images. The mountain images were formed using a similar method based on a 33-by-33 data matrix that is too large to be presented here.

C.2.1 makepair_data.m

```
% Make stereo pair based on data
clear % clear all data
global DATA_MATRIX
DEPTH = 'data_depth'
n = 65; % The size of image.
nn = n+2; % The size of z.
mu = 1; % Optimization parameter

if 1, % Wrinkled surface (hill)
  load data_matrix
  b = 500;
  ps1 = -1; qs1 = 1;
  ps2 = .3; qs2 = .1;
  prefix = ['w',int2str(n),'c'];
```

```
else % Lake (mountain)
  load lake_data
  z0 = -1000;
  DATA_MATRIX = DATA_MATRIX + z0;
  b = 100;
  ps1 = .5; qs1 = .5;
  ps2 = -.5; qs2 = 0;
  prefix = ['lk',int2str(n),'b'];
end
[md,nd] = size(DATA_MATRIX);

f = -1000; % We are working in the same coordinates as the surface for x & y.

% Camera coordinate calibration
z0 = eval([DEPTH,'(0,0)'])
v0 = -b*f/z0/2;
v1 = v0 + b*f/z0/2
v2 = v0 - b*f/z0/2

gamma = f*b/2;
delta = z0/f;

[x,y] = domain2d([1:(md-1)/(nn-1):md]-(md+1)/2,[1:(nd-1)/(nn-1):nd]-(nd+1)/2);
x = x*.9; y = y*.9; % Center 90% of matrix

% Image parameters
deltax = x(1,2)-x(1,1);
params = [f/deltax       % Focal length for unity spacing grid
          b         % baseline distance
          z0             % Calibration depth
          ps1;qs1;ps2;qs2   % Light positions
          (x(2,2)+v1)/deltax-1  % x-calibration as offset from (1,1).
          y(2,2)/deltax-n   % y-calibration as offset from (1,1).
          (x(2,2)+v2)/deltax-1  % x-calibration as offset from (1,1).
          y(2,2)/deltax-n   % y-calibration as offset from (1,1).
          1 % Grid spacing
];
del = z0*(max(x(:))-min(x(:)))/f/(n+1);

z = eval([DEPTH,'(x,y)']);
eqmesh(z,del)
%mesh(z)
title('True Surface')

% Determine depth function for left image (1)

err = 1; m = 0;
z1 = z0*mones(x);
gamma/z0
while (err> 1.e-5)&(m<25),
  ztemp = eval([DEPTH,'(x+v1-gamma./z1-v0,y)']);
  err = sum( (ztemp(:)-z1(:)).^2 )
  z1 = z1 + mu*(ztemp-z1);
  m = m + 1;
end
eqmesh(z1,del)
title('Left Surface')

% determine depth function for right image (2)
err = 1; m = 0;
z2 = z0*mones(x);
while (err>1.e-5)&(m<25),
  ztemp = eval([DEPTH,'(x+v2+gamma./z2-v0,y)']);
  err = sum( (ztemp(:)-z2(:)).^2 )
  z2 = z2 + mu*(ztemp-z2);
  m = m + 1;
end
eqmesh(z2,del)
title('Right Surface')

hx = [-1 0 1]/2;
hy = [1;0;-1]/2;
rows = 2:n+1;
cols = 2:n+1;

% Right Image
zx = filter2d(z1,hx,'resize')/deltax;
zy = filter2d(z1,hy,'resize')/deltax;
p1 = f*zx(rows,:)./((x(rows,cols)+v1).*zx(rows,:)+z1(rows,cols));
q1 = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z1(rows,cols));
E1 = rmap(p1,q1,ps1,qs1);

```

```
% Left Image
zx = filter2d(z2,hx,'resize')/deltax;
zy = filter2d(z2,hy,'resize')/deltax;
p2 = f*zx(rows,:)./((x(rows,cols)+v2).*zx(rows,:)+z2(rows,cols));
q2 = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z2(rows,cols));
E2 = rmap(p2,q2,ps2,qs2);

% Center Global coordinate system
zx = filter2d(z,hx,'resize')/deltax;
zy = filter2d(z,hy,'resize')/deltax;
p = f*zx(rows,:)./(x(rows,cols).*zx(rows,:)+z(rows,cols));
q = f*zy(:,cols)./(y(rows,cols).*zy(:,cols)+z(rows,cols));
E = rmap(p,q,ps2,qs2);

clg,subplot(221)
subplot(121), eqmesh(z1,del)
subplot(122), eqmesh(z2,del)
subplot(111), title('Image contours')

mimage([E1,zeros(n,1),E2],[0 1])

clear p1 q1 p2 q2 zx zy p q E gamma err m n ztemp
clear ps1 ps2 qs1 qs2 z0 b0 b f delta hx hy rows cols v1 v2
clear mu v0 x y md nd nn deltax

ztrue = z;

```

C.2.2 data_depth.m

```
function [z,rho]=depth(x,y)
%DATA_DEPTH  Depth function based on data matrix.
%
%   Z = DATA_DEPTH(X,Y).  X and Y must be of the same size.
%
%   [Z,RHO] = DATA_DEPTH(X,Y) returns albedo also.
%
%   Relies on the global DATA_MATRIX to define surface.
%
%   For DATA_MATRIX m-by-n, values in X must be between -(n-1)/2 and (n-1)/2,
% values in Y must be between -(m-1)/2 and (m-1)/2.

%global DATA_MATRIX
[m,n] = size(DATA_MATRIX);
[xx,yy] = domain2d(1:n,1:m);

x = x + (m+1)/2; y = y + (n+1)/2;

d = find(floor(x)<1); x(d) = ones(length(d),1);
d = find(x>n); x(d) = n*ones(length(d),1);
d = find(floor(y)<1); y(d) = ones(length(d),1);
d = find(y>m); y(d) = m*ones(length(d),1);

z = bicubic(xx,yy,DATA_MATRIX,x,y);
%z = blinear(xx,yy,DATA_MATRIX,x,y);

d = find(isnan(z));
if length(d)>0, keyboard, end
if length(d)>0, z(d) = ones(length(d),1)*min(DATA_MATRIX(:)); end

if nargout>1,
%d = find(abs(x-y)<1);
z0 = min(z(:));
d = find((z>.5+z0) & (z<.7+z0));
rho = mones(z);
rho(d) = 0.7*mones(d);
end
```

Bibliography

Aloimonos, J. Y. and A. Basu (1988). Combining information in low-level vision. In *Image Understanding Workshop*, pp. 862–906. DARPA, April 1988.

Belknap, R., E. Riseman, and A. Hanson (1986). The information fusion problem and rule-based hypotheses applied to complex aggregations of image events. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 227–234. IEEE, 1986.

Bogler, P. L. (1987). Shafer-dempster reasoning with applications to multisensor target identification systems. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, No. 6, pp. 968–977, Nov/Dec 1987.

Brandt, A. and N. Dinar (1979). Multigrid solutions to elliptic flow problems. In Parter, S. V., editor, *Numerical Methods for Partial Differential Equations*, pp. 53–145. Academic Press, 1979.

Brown, J. R., D. Bergondy, and S. Archer (1991). Comparison of neural network classifiers to quadratic classifiers for sensor fusion. In *Applications of Artificial Neural Networks II*, pp. 539–543. SPIE, 1991.

Courant, R. and D. Hilbert (1962). *Methods of Mathematical Physics, Volume I.* Wiley, New York, NY, 1962.

Davis, P. A. and A. S. McEwen (1984). Photoclinometry: Analysis of inherent errors and implications for topographic measurements. In *Lunar and Planetary Science Conference XV*, pp. 194–195, 1984.

Davis, P. A. and L. A. Soderblom (1984). Modeling crater topography and albedo from monoscopic viking orbiter images: 1. methodology. *Journal of Geophysical Research*, Vol. 89, No. B11, pp. 9449–9457, Oct 10 1984.

Gelb, A. (1974). *Applied Optimal Estimation.* The MIT Press, Cambridge, MA, 1974.

Geman, S. and D. Geman (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721–741, Nov 1984.

Gennert, M. A. (1987). *A Computational Framework for Understanding Problems in Stereo Vision.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Sept 1987.

Grimson, W. E. L. (1979). *From Images to Surfaces: A Computational Study of the Human Visual System.* PhD thesis, M.I.T., 1979.

Grimson, W. E. L. (1984). Binocular shading and visual surface reconstruction. *Computer Vision, Graphics, and Image Processing,* Vol. 28, pp. 19–43, 1984.

Hadamard, J. (1923). *Lectures on the Cauchy Prolem in Linear Partial Differential Equations.* Yale University Press, New Haven, CT, 1923.

Harmon, S. Y., G. L. Bianchini, and B. E. Pinz (1986). Sensor data fusion through a distributed blackboard. In *International Conference on Robotics and Automation,* pp. 1449–1454. IEEE, 1986.

Hartt, K. and M. Carlotto (1989). A method for shape-from-shading using multiple images acquired under different viewing and lighting conditions. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition,* 1989.

Heipke, C. (1992). Integration of digital image matching and multi image shape from shading. In *XVII Congress of the International Society for Photogrammetry and Remote Sensing,* Washington, DC, August 2-14 1992.

Helfenstein, P., J. Hillier, C. Weitz, and J. Veverka (1991). Oberon: Color photometry from voyager and its geological implications. *Icarus,* Vol. 90, pp. 14–29, 1991.

Horn, B. K. P. and M. J. Brooks (1986). The variational approach to shape from shading. *Computer Vision, Graphics, and Image Processing,* Vol. 33, No. 2, pp. 174–208, Feb 1986.

Horn, B. K. P. and B. G. Schunk (1981). Determining optical flow. *Artificial Intelligence,* Vol. 17, pp. 185–203, 1981.

Horn, B. K. P. (1986). *Robot Vision.* The MIT Press, Cambridge,MA, 1986.

Horn, B. K. P. (1989). Height and gradient from shading. A.I. Memo 1105, MIT Artificial Intelligence Laboratory, May 1989.

Hu, G. and N. Shrikhande (1990). Fusion of gray scale and light striping in 2-d feature extraction. In *Third International Conference on Industrial and Engineering Applications of AI and Expert Systems,* pp. 156–162. ACM, 1990.

Hung, Y.-P., D. B. Cooper, and B. Cernuschi-Frias (1988). Bayesian estimation of 3-d surfaces from a sequence of images. In *International Conference on Robotics and Automation,* pp. 906–911. IEEE, 1988.

Ikeuchi, K. and B. K. P. Horn (1981). Numerical shape from shading and occluding boundaries. *Artificial Intelligence*, Vol. 17, pp. 141–184, 1981.

Keys, R. G. (1981). Cubic convolution interpolation for digital image processing. *IEEE Transactions of Accoustics, Speech, and Signal Processing.* Vol. 29, No. 6, pp. 1153–1160, Dec 1981.

Kirk, R. L. (1987). *A Fast Finite-Element Algorithm for Two-Dimensional Photoclinometry.* PhD thesis, Division of Geological and Planetary Sciences, California Institute of Technology, Pasedena, CA, 1987.

Leclerc, Y. G. and A. F. Bobick (1991). The direct computation of height from shading. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 552–558. IEEE, June 3–6 19[illegible].

Lee, R. H. (1990). Multi-sensor image segmentation algorithms. In *Sensor Fusion III*, pp. 11–17. SPIE, 1990.

Matthies, L. and A. Elfes (1988). Integration of sonar and stereo range data using a grid-based representation. In *International Conference on Robotics and Automation*, pp. 727–733. IEEE, 1988.

McEwen, A. S. (1985). Albedo and topography of ius chasma, mars. *Lunar and Planetary Science Convention XVI*, Vol. , pp. 528–529, 1985.

McEwen, A. S. (1991). Photometric functions for photoclinometry and other applications. *Icarus*, Vol. 92, pp. 298–311, 1991.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equations of state calculations by fast computing machines. *Journal of Chemisty and Physics*, Vol. 21, No. 6, pp. 1087–1092, June 1953.

Mitiche, A. and J. K. Aggarwal (1986). Multiple sensor integration/fusion through image processing: A review. *Optical Engineering*, Vol. 25, No. 3, pp. 380–386, March 1986.

Moerdler, M. L. and T. E. Boult (1988). The integration from stereo and multiple shape-from-texture cues. In *Image Understanding Workshop*, pp. 786–793. DARPA, April 1988.

Moerdler, M. L. and J. R. Kender (1987). An approach to the fusion of multiple shape from texture algorithms. In *Spatial Reasoning and Multi-Sensor Fusion*, pp. 272–281, 1987.

Nandhakumar, N. and J. K. Aggarwal (1988). Thermal and visual information fusion for outdoor scene perception. In *International Conference on Robotics and Automation*, pp. 1306–1308, 1988.

Negahdaripour, S. and B. K. P. Horn (1985). Determining 3-d motion of planar objects from image brightness measurements. In *International Joint Conference on Artificial Intelligence*, pp. 898–901. Los Angeles, CA, August 1985.

Pau, L. (1989). Knowledge representation approaches in sensor fusion. *Automatica*. Vol. 25, No. 2, pp. 207–214, 1989.

Richardson, J. M. and K. A. Marsh (1988). Fusion of multisensor data. *The International Journal of Robotics Research*, Vol. 7, No. 6, pp. 78–96, Dec 1988. Good formulation of Baysian estimation.

Saxberg, B. V. H. (1989). A modern differential goemetric approach to shape from shading. TR 1117, M.I.T. Artificial Intelligence Laboratory, 1989.

Shaw, S. W., R. J. P. deFigeuiredo, and K. Krishen (1988). Fusion of radar and optical sensors for space robotic vision. In *International Conference on Robotics and Automation*, pp. 1842-1846. IEEE, 1988.

Szeliski, R. (1990). Fast surface interpolation using hierarchical basis function. *IEEE PAMI*, Vol. 12, No. 6, pp. 513–528, June 1990.

Szeliski, R. (1991). Fast shape from shading. *Computer Vision, Graphics, and Image Processing*, Vol. 53, No. 2, pp. 129–153, March 1991.

Terzopoulos, D. (1984). Efficient multiresolution algorithms for computing lightness, shape from shading, and optical flow. In *Proc. of the Fourth National Conference on Artificial Intelligence*, pp. 314–317, Austin, TX, August 1984.

Thomopoulos, S. C. A. (1989). Sensor integration and data fusion. In *Sensor Fusion II: Human and Machine Strategies*, pp. 178-191. SPIE, 1989.

Tikhonov, A. N. and V. Y. Arsenin (1977). *Solutions of Ill-Posed Problems*. Winston & Sons, Washington, D.C., 1977.

Van Hove, P. L. and M. J. Carlotto (1986). An iterative multi-resolution shape-from-shading algorithm and its application to planetary mapping. In *International Geoscience and Remote Sensing Symposium*, pp. 1011–1016, 1986.

Waxman, A. M. and J. H. Duncan (1986). Binocular image flows: Steps toward stereo-motion fusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, pp. 715–729, Nov 1986.

Wildey, R. L. (1973). Theoretical autophtogrammetry: I. the method of photometric potential. *Modern Geology*, Vol. 4, pp. 209-215, 1973.

Wildey, R. L. (1975). Generalized photoclinometry for mariner 9. *Icarus*, Vol. 25, pp. 613–626, 1975.

Wilson, L., J. S. Hampton, and H. C. Balen (1985). Photoclinometry of terrestrial and planetary surfaces. In *Lunar and Planetary Science Conference XVI*, pp. 912-913, 1985.

Woodham, R. J. (1980). Photometric method for determining surface orientation from multiple images. *Optical Engineering*, Vol. 19, No. 1, pp. 139-144, Jan/Feb 1980.